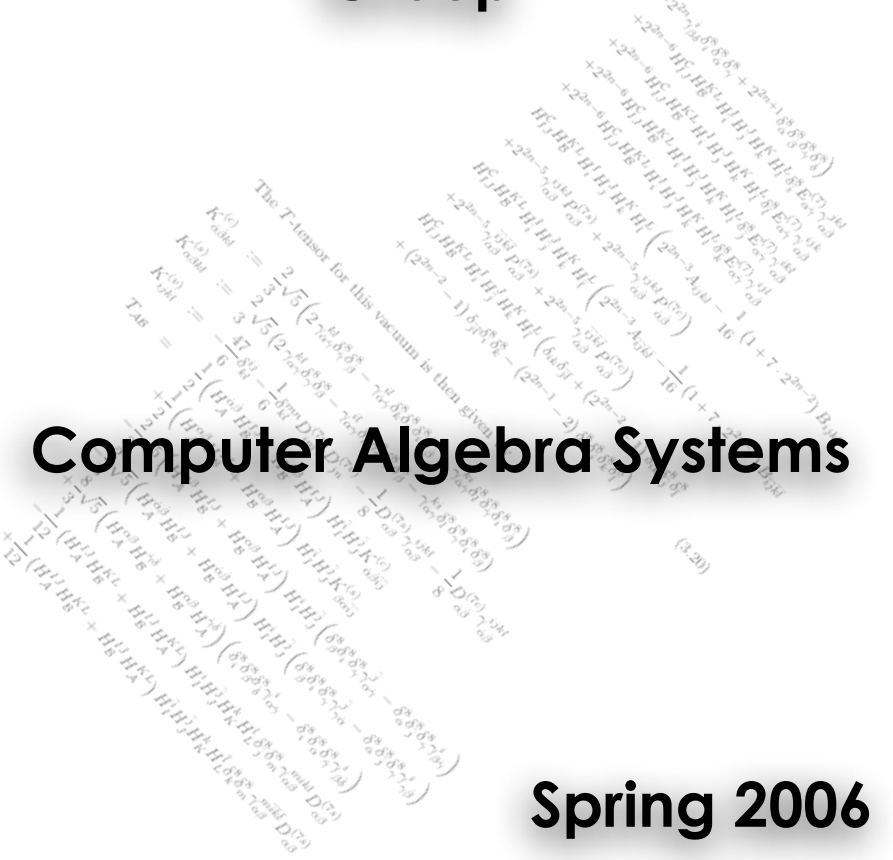
Institute *of* **Physics**

Newsletter

of the

Computational Physics Group



Computer Algebra Systems

Spring 2006

MEMBERS OF THE COMMITTEE

Roger Barrett	R.Barrett@surrey.ac.uk
Allan Boardman (treasurer, secretary)	a.d.boardman@salford.ac.uk
Peter Borcherds (chairman)	p.h.borcherds@birmingham.ac.uk
Alan DuSautoy	alan.dusautoy@npl.co.uk
Hans Fangohr (newsletter)	h.fangohr@soton.ac.uk
Andrew Horsfield	a.horsfield@ucl.ac.uk
Dave Hutchings	d.hutchings@elec.gla.ac.uk
Geraint Lewis (travel bursaries)	dg.lewis@physics.org
Jim McNiff	jemc@tinyonline.co.uk
Ian Morrison	i.morrison@salford.ac.uk
Massimo Noro	Massimo.Noro@Unilever.com
Matt Probert	mijp1@york.ac.uk
Michael Sleigh	Michael.Sleigh@awe.co.uk

Our web page can be found here: http://groups.iop.org/CP/

Comments about the Newsletter should be sent to Hans Fangohr.

Front cover picture: equations taken from http://xxx.arxiv.org/abs/hep-th/0305176 (page 42).

Newsletter Contents

Quo Vadis, Computer Algebra?

Dr Thomas Fischbacher, University of Southampton

Nowadays, the term "Scientific Computing" is used almost synonymously with "numerical simulation on supercomputers". We nevertheless also observe a gradual shift towards a broader, and perhaps much more adequate, interpretation of that term, for many important recent applications of computing to science have a much stronger emphasis on algorithmic aspects than on number crunching. This is especially true in bioinformatics, perturbative quantum field theory, or computational linguistics. Indeed, the otherwise ubiquitous Teraflop is not an adequate measure for most computations in these disciplines: if at all, only very few floatingpoint operations are involved, and they usually do not play a prominent role.

From the perspective of physics, the second largest class (after number crunching) of computer applications is the realm of 'symbolic algebra'. As in many other domains, there are a few well-known programs on the market that shape peoples' idea of symbolic algebra more than anything else – the most important ones here being Maple (by Waterloo Software) and Mathematica (by Wolfram Inc.).

What is Computer Algebra?

This circumstance alone is a good reason to ask for a proper definition of the term 'symbolic algebra' that is not tied to the name of a few superficially similar contemporary programs which nevertheless work very differently under the hood. Unfortunately, finding a good definition turns out to be a difficult task, for on the one hand, there are many systems that are based on symbolic manipulation and re-writing ideas which one usually would not at all consider to be symbolic algebra packages, among them e.g. Donald Knuth's typesetting system TEX– or, for that matter, even any FORTRAN or C compiler. On the other hand, the capability to do a certain minimal set of general formula manipulations cannot be a criterion either, as there are special-purpose packages such as `LiE`[lie] that do some important and complicated symbolic mathematical computations, but do not even know about some of the most fundamental types of expressions – e.g. rational functions in one variable. It is furthermore noteworthy that by now, stock computer algebra packages also seem to have become somewhat popular to address problems which involve barely any advanced symbolic manipulation, but are mostly graph-theoretically or combinatorically minded, dealing e.g. with finite permutations, their products and inverses only. This may be regarded as a

strong indication that a computer algebra system must be more than just a tool for manipulating mathematical terms.

A historical perspective

Facing these difficulties in a head-on approach towards a proper demarcation, it may be enlightening trying to understand the history of symbolic computation. Just as with the history of computing itself, there is some arbitrariness involved in what to count as the first conception and realization of the idea. In the same spirit that we will not consider Leibniz' 1673 mechanical step reckoner (which could add, multiply, subtract, divide, and compute square roots) as a computer, we will neither consider the 'bomba kryptologiczna' – an early electro-mechanical device to decipher the German's ENIGMA code during World War II – nor the 1953 theses by Kahrimanian and by Nolan on automatic differentiation as the dawn of symbolic algebra, even if it cannot be denied that some important ideas date back to this time, or even earlier. What should be considered a quantum leap in our understanding of symbolic computation was John McCarthy's 1960 conception of a general scheme to conveniently express arbitrary manipulations of formal expressions in a way that can also be understood by a computer [car]. McCarthy's key ideas were that (1) symbolic manipulations such as the differentiation of a term can and should be modeled through recursive functions, conditional expressions, and hierarchical data structures, (2) it should be possible to use functions as arguments to functions, and (3) the mathematical simplicity and beauty of recursive definitions should not be sacrificed on the altar of machine efficiency, in particular not to in-place updates (i.e. replacing the memory representation of a term by that of its derivative in order to save memory cells).

These three points need closer investigation: Concerning the first, McCarthy did not consider automatic differentiation as an important problem per se (we already have seen that this was solved somewhat earlier), but rather a prime example exhibiting most of the relevant characteristics of a typical symbolic manipulation. The second idea of allowing functions as arguments to functions is far more subtle than one would guess at first, due to a double meaning of the term 'function'. This is not so much about passing a term like $x^4 + x^2 \sin x$ to a differentiation function, but about generic functions which are parametrized by other functions. For example, differentiating a sum which is represented as a list of summands can be defined elegantly as applying 'the differentiation procedure' to every summand (and collecting the results in a suitable way). As the general idea of mapping a list to a list by applying some transformation to every element is highly useful in itself, the differentiation procedure therefore should re-

cursively pass the differentiation procedure to the elementwise-mapping-over-list procedure when differentiating a sum. This actually sounds much more complicated than it is, just because the English language is not very well suited for talking about such abstract concepts (as this is hardly ever necessary in day-to-day conversation). It is, however, highly instructive to have a look at a formal description of a toy example in a notation that is closer to the conventional mathematical symbolism than McCarthy's original notation, and hence presumably may also be understood by the non-initiated. This is given in Figure 1.

Note that the example term on which we perform manipulation is just an instance of some recursive data type. As it stands, it clearly is not a function in the usual Dirichlet sense of a mapping of values from a domain to a range: we did not define how to evaluate it on numbers. On the other hand, the `derive` function may be regarded as being such a mapping – even a computable one – with values belonging to the `Polynomial` data type in its domain. (This involves some additional subtlety, but we will not delve deeper here.) In addition, the `derive` function also is used recursively inside the `derive` function, both as a function (to recursively compute the derivatives in products) as well as an argument to another function – the `map` function that applies `derive` to every summand in a sum, that is. There is a mathematical theory that deals with this way of applying functions to functions which has quite a long history of its own, which is known under the names 'Lambda Calculus' or 'Combinatory Logic'. It is furthermore noteworthy that `derive` is passed around in a 'black box' fashion to `map`, as the recipient only can evaluate this function where needed, but has no need (and, in fact, no means) to introspect and analyze the definition of `derive`.

This single example also shows us the importance of McCarthy's third key point: our definition of the `derive` function can be regarded as a purely mathematical recursive specification that does not at all deal with questions such as where to allocate memory to hold the values, when to reclaim memory and when to do in-place modifications. As it turns out, quite many algorithmically sophisticated applications benefit greatly from the separation of machine-level details such as memory management from the actual problem.

An important observation can be made here: looking at the history of complex systems as different as the Mathematica symbolic algebra package, the Netscape web browser, the GNU C compiler, and many more, one sees a clear evolution in their memory management subsystems that – sometimes after a certain amount of meandering and painful discoveries – invariably gravitate towards a common core that can be implemented in a general and application-independent way, even in the form of a library [bdw].

```
> import Ratio -- we use rational numbers

> data Polynomial =           -- a polynomial is either
>   Constant (Ratio Integer)  -- a constant rational number
>   | X                       -- or the variable X
>   | Sum [Polynomial]        -- or a sum of polynomials
>   | Product [Polynomial]    -- or a product and we use the
>   deriving (Show)           -- default way to print polynomials.

The definition of the derivative:

> derive (Constant n) = Constant 0

> derive X = Constant 1

> derive (Sum summands) = Sum (map derive summands)

> derive (Product []) = Constant 0

For a one-factor product, we just derive the factor

> derive (Product [f]) = derive f

> derive (Product (factor1:rest_factors)) =
>     Sum [Product ((derive factor1):rest_factors),
>          Product (factor1:[derive (Product rest_factors)])]

An experiment:

> some_example_polynomial = Sum [X,Product [Constant 2,X,X]]

] derive some_example_polynomial
Gives:

Sum [Constant (1 % 1),
     Sum [Product [Constant (0 % 1),X,X],
          Product [Constant (2 % 1),
          Sum [Product [Constant (1 % 1),X],
               Product [X,Constant (1 % 1)]]]]]

This is the expression 1+(0*X*X + 2*(1*X + X*1)).

For the sake of brevity, simplification is not addressed in this
example.
```

Figure 1: A "toy" implementation of a differentiating function. (This actually is a valid, executable program in literate Haskell.)

On LISP

McCarthy's ideas led to the design and implementation of LISP – which nowadays is considered a 'programming language', but rather should be viewed as a minimal core for specifying any kind of symbolic transformation in a way that can be understood by a computer. Curiously, as the execution of a LISP program again is just a special kind of symbolic transformation, and due to the minimality and generality of the LISP system, it is almost a trivial exercise to implement a LISP interpreter in LISP. More importantly, it is very easy to write LISP code that generates LISP code and to use this feature to introduce new linguistic devices into the language. This may at first sound like employing self-modifying code or conventional macro programming, but this is a misleading thought. For one, syntactic extensions are performed in a purely downstream fashion, so no piece of code will ever 'modify itself' in a properly written LISP program, and secondly, the transformations take place at the level of a tree structure representation of the code, not at the level of isolated tokens, as is the case e.g. with the C preprocessor. The major difference is that full contextual information is present in the syntactic tree, but not in a small piece of the token stream. At present, we see a re-invention of these ideas in the form of using XML to represent structured data – such as program code – and performing generic transformations on these trees.

The essence of LISP presumably can be summarized in a nutshell as follows: LISP is a minimal extract of those generic common core capabilities that have to be present in virtually all complex applications and usually are difficult to implement in a proper way. (Including in particular flexible dynamical memory management and programmability). As LISP tries to be minimal, it must be syntax-agnostic and not impose any notational restrictions. This is achieved by working not with expressions such as `x=x+1`, but with representations of program code in the form that usually is produced by a *parser*, like `(set! x (+ x 1))`. This way, one may put any syntax one wants on top of LISP and benefit from all the hard work (such as memory management, interpretation, and even compilation to fast machine code) already having been done by the LISP implementors. This minimal core system must, however, provide one additional feature: the capability to be used as its own code transformation language, so that LISP may be used to progressively extend the initially minimal LISP core with new capabilities, without being bound by insurmountable syntactical restrictions. Thus, it becomes possible to model the structure of the language towards the desired application, rather than having to forcefully fit a potentially ingenious new idea into an inflexible pre-existing framework. Just as every discipline has its own

jargon, occasionally including highly specialized formalism (as in Chemistry or Mathematics), the core of LISP just contains everything that is needed to implement any kind of specialized jargon for human-computer interaction, and nothing more. This unprecedented flexibility, limited only by one's fantasy and creativity, usually requires some direct experience before it can be fully appreciated.

Pioneering Systems and their Children

Early LISP-based computer algebra systems from the 1960's and 1970's included REDUCE (by A. Hearn) as well as MACSYMA (developed at MIT). The latter can be considered the mother of most of the more recent symbolic algebra packages. An unrelated but nevertheless important line of development was started in particle physics by M. Veltman, who implemented the 'Schoonschip' program (in assembly language) to do a certain kind of non-commutative operator algebra [shp]. Schoonschip especially simplifies the (otherwise often quite tedious) task of working with Clifford algebras (also known as Dirac or Gamma matrices), which are indispensable for quantitative computations with relativistic fermionic quantum fields. Schoonschip played an important role in the work that earned t'Hooft and Veltman the Nobel Prize in Physics in 1999 for the renormalization of the Standard Model. A successor of Schoonschip – Jos Vermaseren's FORM – is designed along the same principles as an efficiency-oriented but not general-purpose term manipulation system that can crunch large expressions and is still very popular in the particle physics community. These packages are not based on McCarthy's ideas, and do not strive for the full flexibility of a general-purpose programming environment.

Both Maple and Mathematica can be regarded as being inspired by MACSYMA, but with slightly different motivation. As efficient compilation of LISP to machine code is a difficult task that required a lot of research effort (and therefore time) until key concepts were understood properly, the early LISP systems earned a reputation of being horrendously inefficient in terms of memory and CPU utilization. To a certain extent, this is still a widely held (but nowadays mostly unfounded) belief. There was another reason why LISP had fallen from grace, namely that striking early successes in the field of Artificial Intelligence led to high expectations that turned out to be unsatisfiable. Presumably for these reasons, both Maple and Mathematica were implemented in a variant of the C programming language, and designed as interactive interpreter systems. While Maple more or less is a straightforward implementation of an interpreter language including data types for symbolic mathematics, Mathematica's most distinguishing feature is to rely on a somewhat unusual rule-driven term eval-

```
> coords:=[x,y,z]:
> for x in coords do lprint([x,coords]) od;
[x, [x, y, z]]
[y, [y, y, z]]
[z, [z, y, z]]
> coords;
          [z, y, z]
```

Figure 2: A potential Maple trap.

uation model that unfortunately now is known to exhibit some serious design flaws [fat]. But also with Maple, certain questions appear to have been resolved in a somewhat unsatisfactory ad-hoc way, among them the improper separation of the 'object level' (where mathematical terms are represented) from the 'implementation level' (where term manipulations live). Figure 2 shows a simple example that demonstrates one non-obvious pitfall concerning issues of proper variable scoping and unexpected modification through side effects.

It is furthermore interesting to note how both these systems cannot escape providing some of the fundamental primitives of LISP – such as anonymous functions – at the interpretive level.

There are two further members of the MACSYMA family that deserve special mentioning: MuPAD – which was developed at the University of Paderborn and has a close resemblance to Maple in many aspects – as well as MAXIMA, which is a branch of the original MACSYMA code that is available under a free license (the GNU GPL). MAXIMA is especially interesting because – due to its history – it is implemented not as a monolithic system, but as a Common LISP library. Hence, it benefits from the dramatic progress in LISP compilation that happened during the last decades – independent from its own evolution. Furthermore, it can be used on top of a variety of LISP systems, and also can be combined with a wide range of other LISP libraries. While its scope may be more limited than that of commercial systems, it definitely is an interesting alternative if it comes to extending some complex software package – maybe even a web-based application – with algebraic capabilities.

It should not be forgotten that – due to its inherent flexibility and extensibility – even a bare LISP system by itself sometimes provides a good basis to perform a symbolic computation. This is especially the case if the problem at hand is conceptually simple, and speed is so important that it can benefit from cleverly tailored internal term representations and compilation to fast machine code.

An Outlook

As the fields of application of popular computer algebra systems become ever broader, two things are gradually realized: firstly, computations involving mathematical expressions at the symbolic level is not something that can be considered in isolation, but experiences a lot of cross-fertilization even at the lowest levels: on the one hand, the proper algorithms to implement tensor algebra are just the same ones as those being used in relational databases (a contracting tensor product being quite equivalent to a SQL table JOIN), on the other hand, a finite element package working with general unstructured meshes benefits from the ability to integrate polynomial functions over simplices analytically. In situations like these, independently of whether computer algebra receives or provides expertise with respect to other pieces of a program, benefits are greatest if these components can be integrated as tightly as possible. Hence, there is a certain pragmatical urge today to go back from monolithic systems to computer algebra systems implemented as libraries. Conceptually, this may also be regarded as a hint towards the necessity of a much broader interpretation of 'computer algebra' which also considers data such as program code as being 'terms', and hence would eventually include all kinds of symbolic transformations – in particular, compilers. Actually, it is hard to deny that a compiler has to do some stringent reasoning on allowed transformations of symbolic expressions that are defined in a mathematically sound way (because they adhere to some formal grammar), and even has to employ knowledge of fundamental laws of mathematical operations. It certainly would be somewhat unjustified to deny this kind of mathematical reasoning the right to be called 'mathematics', just because it does not necessarily involve symbols denoting numbers. Aside from these considerations, we also see a general broadening of research activity in the field of computer-aided mathematics, which also includes research on topics such as semi-automatic proof assistants.

Maybe, the best answer to the question 'what is computer algebra' that one can give is: It is a loosely defined field that nowadays encompasses all kinds of computations on and transformations of mathematical formulae, but still is based on an artificially narrowed down definition of a 'formula', which does not yet include all symbolic expressions that are used in mathematical reasoning. Due to this restriction, it is a true subset of the larger discipline of symbolic computation, which deals chiefly with accurate mathematics and does not involve numerical approximations.

References

[bdw] Boehm-Demers-Weiser Garbage Collecting malloc() library:
http://www.hpl.hp.com/personal/Hans_Boehm/gc/

[fat] http://citeseer.ist.psu.edu/fateman93review.html

[lie] http://wwwmathlabo.univ-poitiers.fr/ maavl/LiE/

[car] Recursive functions of symbolic expressions and their computation by machine (Part I), on the web at:
http://www-formal.stanford.edu/jmc/recursive.html

[shp] Schoonschip can be obtained from
http://www-personal.umich.edu/ williams/archive/schoonschip/index.html

Dr Thomas Fischbacher is currently working in the School of Engineering Sciences at the University of Southampton and can be contacted by email at `t.fischbacher@soton.ac.uk`.

Computational Physics Group News

IoP Computational Physics Group - Student Travel Award

Organised by: Geraint Lewis

The Computational Physics Group (CPG) of the Institute of Physics (IoP) is pleased to invite requests for partial financial support towards the cost of attending scientific meetings relevant to the Group's scope of activity, as outlined on our web page: http://groups.iop.org/CP/. The aim of the scheme is to help stimulate the career development of young scientists working in computational physics to become future leaders in the field.

To be eligible the applicant should:

- be a full time PhD student;
- provide evidence of acceptance of a presentation (oral or poster) at the meeting in question;
- give an itemised estimate of cost of attendance;
- provide a letter of support from their project supervisor which:
  - ▷ confirms the applicant's PhD student status;
  - ▷ explains the relevance of the meeting;
  - ▷ details the source of the additional funds necessary to attend the meeting.

Applications are invited at any stage in a given year, but will be reviewed by the CPG Committee on a quarterly basis (1st March, 1st June, 1st September, 1st December). Successful applicants will be notified as soon as possible thereafter. Candidates are advised to make their submissions well in advance of the meeting they wish to attend. The maximum support available to any applicant will be 200. The CPG's decision regarding financial support and its level will be final and non-negotiable in all cases.

Successful applicants will be asked to provide a short written report of the meeting suitable for publication in the CPG Newsletter.

For further details, please contact:

Dr D.G.Lewis
Department of Medical Physics
Velindre Hospital
Cardiff CF14 2TL
e-mail: dg.lewisphysics.org
Tel: 029 2019 6192

The 4^{th} Annual Computational Physics Thesis Prize

The Committee of the Institute of Physics Computational Group has endowed two annual prizes. £500 will be awarded to the author of the PhD thesis that contributes most strongly to the advancement of computational physics. Two runners-up will receive £250. There will be free group membership for 2006 for *all* entrants. The Committee will select the recipients and its remit will be very broad, in order to capture a broad spectrum of modelling activity.

- The deadline for applications is July 31^{st}, 2006. The competition is open to all students whose PhD examination has taken place in 2005.
- The submission format is a 4 page (A4) abstract together with a citation (max. 500 words) from the PhD supervisor.
- The submission address is:
  Dr M Probert
  Department of Physics
  University of York
  York, YO10 5DD

Applicants must have carried out their thesis work at a University in the United Kingdom or the Republic of Ireland.

The following were the winners of 2004 competition:

- 1st place: Neil Drummond & Nick Parker
- 3rd place: Karen Cairns, Natalia Martsinovich & Arash Mostofi

Reports on meetings

A gentle introduction to biological modelling

Organised by: Andrew Horsfield

The Computational Physics Group of the Institute of Physics held a one-day workshop on the modelling of biological systems on Thursday, 15 September 2005 at the Institute of Physics, 76 Portland Place, London. This meeting was targeted at physicists who have little or no knowledge about biological modelling but who wish to know more. There were four talks, each one hour long. The abstracts are given below.

A special thanks goes to the BBSRC who sponsored this meeting. A representative (Adam Staines) gave a short presentation on what the BBSRC supports, and how to apply for funding.

The meeting was attended by over 50 people.

An Introduction to Cell Biology

Jenny Owens

With the exception of viruses, all living organisms are based on a specific structural unit, the cell. Despite their complexity, cells are formed from a relatively small number of elements, which combine to form a relatively small number of types of organic molecules. Based on the arrangement of their genetic material, cells fall into two groups: the smaller, simpler Prokaryotes and the more complex Eukaryotes. Advances in electron microscopy techniques over the last 50 years have led to a huge increase in knowledge of the structural organisation within cells. There is remarkable similarity between cells of different origin as to the structure and functions of their sub-cellular organelles. The genetic material, which controls the synthesis of proteins, is DNA. All cells are surrounded by a lipid/protein bilayer that controls movement of substances in and out of the cell. A protective wall surrounds plant and prokaryotic cells. In eukaryotes the internal body of the cell, the cytoplasm, is partitioned and supported by many layers of membrane folded into compartments, the endoplasmic reticulum. This both separates enzyme systems and supports other organelles. Ribosomes, responsible for the synthesis of proteins, attach to the surface of the e.r., and Golgi apparatus, which processes and packages protein prior to its secretion from the cell is a specialised region of smooth e.r.. Energy is made available through aerobic respiration in mitochondria (all cells) and photosyntheis in chloroplasts

(plant cells). Lysosomes contain enzymes that will be released at apoptosis, an important process in the replacement of ageing cells.

Biology of Proteins and DNA

Sarah Harris

Biological macromolecules (such as DNA and proteins) mediate all cellular processes at the atomic level. This seminar will provide a very basic introduction to the structure and function of biological molecules from a physicists point of view, including the thermodynamics of molecular recognition, the use of DNA processing motors to access and read the genetic code and an overview of the role of DNA damage and repair. An emphasis will be placed on identifying biological processes that are currently poor

Modelling Proteins and Membranes

Syma Khalid

Biological membranes are highly selective barriers that are crucial to the life of the cell. The plasma membrane encloses all cells and maintains the essential differences between the cytosol and the extracellular environment. Eukaryotic cells contain additional elaborate systems of internal membranes which create compartments within the cell. Whilst small, organic solvents are able to diffuse through the membrane, the transport of larger charged species is facilitated by proteins. In all cells, the plasma membrane contains proteins that act as sensors of external signals. Sensory proteins allow the cell to modify its behaviour in response to external stimuli.

Membrane proteins play a key role in the function and structural integrity of the membranes in which they reside. They comprise 30% of open reading frames and yet we know the x-ray structures of only 40 membrane proteins (they are notoriously difficult to crystallise). It is therefore imperative that we extract the maximum possible information from the available structures. In particular it is of importance to explore the dynamical behaviour of membrane proteins since this is a key step in relating structure to biological function. Computational techniques such as homology modelling and Molecular Dynamics (MD) simulations provide an ideal way to study the dynamics of membrane proteins and have been widely employed for this purpose.

In this lecture, I will cover the background and methods for molecular modelling of membrane proteins based on structures of homologues and molecular dynamics simulations of membrane proteins in a lipid bilayer environment.

Reference:

Molecular dynamics simulations of biomolecules, M. Karplus, J.A. McCammon, Nature Structural Biology 9, 646-652, 788 (2002)

Biology in the 21st Century: New Challenges for a Data Intensive Science

Geoff Barton

Over the last 20 years, research in biology and in particular molecular biology, has moved from a "cottage industry" of individuals, to a team and data intensive science that relies heavily on computation and effective data management for interpretation. The field of Bioinformatics addresses the wide-ranging issues of organisation, analysis and interpretation of the wealth of biological data now available. In this talk I will summarise the different techniques and technologies that are providing this new flood of data ranging from DNA sequences through various "post-genomic" techniques such as microarray and protein mass-spec and highlight some of challenges these present for bioinformatics research.

Computer Languages for Scientific Computing

Organised by: Matt Probert

The Computational Physics Group held a 1 day meeting at the Institute of Physics, 76 Portland Place, London on 22 April 2005. The aim of the meeting was to broaden our general knowledge of computer languages and to better equip the audience to choose the right tool for the right job. With this in mind, the day consisted of 6 talks on different popular computer languages, and 1 talk on tools, followed by a general question-and-answer session with a panel of all the speakers. The languages examined were Fortran90/95, C, C++, Java, Fortran2003 and Python. The speakers were all academics with extensive experience of scientific programming in the appropriate languages, with the exception of the talk on Fortran2003 for which John Reid, one of the main authors of this and previous revisions to the Fortran language, gave us a very useful insight into one of the newest languages available.

The meeting was a great success, with many lively discussions after each talk and also during the lunch and refreshment breaks. The meeting was oversubscribed and late applicants had to be turned away. The audience of almost 50 people, included many postgraduate students and representatives from industry.

Full copies of all the talks given are available on the IoP website at http://conferences.iop.org/COL/

Soft interfaces with hydrodynamic interactions

Organised by: Massimo Noro, Patrick Warren, Wim Briels

The Conference

The conference was held over 29-30 September 2005.

The scope was hydrodynamics for materials properties and behaviour in the SoftComp area. The idea was to take a problem-oriented approach rather than a technique-oriented approach.

One class of problems can be described as microhydrodynamics for colloids, emulsions, and vesicles - in other words rigid and deformable particulate suspensions. The problems cover rheology, sedimentation and creaming, the effects of confining geometries, and the behaviour of particles near walls in attachment, deposition and removal scenarios. The particles are typically 100nm to 1 m, and Brownian motion may or may not be relevant. The flows are at low or vanishing Reynolds number. A characteristic of these problems is the need to capture both long range and short range (lubrication) hydrodynamics.

Another class of problems are hydrodynamic interactions for polymers, wormlike micelles, and membranes. These can be characterised as bead-and-spring models, where the hydrodynamic characteristics of the beads may not need to be precisely specified if all that matters is the long range hydrodynamics.

Another area is that of truly microscopic flow problems such as film breakage, contact line dynamics, and the microscopic origin of friction in polymer and related models. The last area is the challenge of implementing truly multiscale (in space) models, linking microscale to macroscale flows - think of the influence of contact line dynamics on the spreading of an emulsion droplet.

Whilst molecular dynamics seems to be universally accepted for microscale problems, there are diverse techniques for mesoscale problems, each with their own characteristic advantages and disadvantages. Such methods include Stokesian dynamics, Brownian dynamics with full resolution of hydrodynamic interactions, lattice Boltzmann methods, multi-particle collision dynamics (also known as stochastic rotation dynamics, or the Malavanets-Kapral method), dissipative particle dynamics, kinetic Monte-Carlo (Bird's method), smoothed particle hydrodynamics, and possibly many others.

The workshop attracted practitioners in these areas from both inside and outside SoftComp, to discuss the relative merits and drawbacks of the methods in the context of the sort of problems that SoftComp faces, and identify gaps.

The workshop also had a training element, and exposed post-docs and PhD students to the variety of methods that are available.

The Model

The workshop was open to all SoftComp partners, with selected speakers from some expert groups outside SoftComp.

The format was 1 + 1/2 days, starting in morning of the first day (people arrive during previous evening), and finishing after lunch on the second day to give people the chance to travel back.

Conference Program

Format was 12 talks - at least one speaker from each participating group, plus the invited non-SoftComp speakers. Ample time was set aside for discussions. Space was made available for people to display informal poster. Speakers list:

- Noro (Unilever)
- Gompper (U. Julich - Germany)
- Padding (U. Twente - Netherlands)
- Pagonabarraga (U. Tarragona - Spain)
- Pastorino (U. Mainz - Germany)
- Boek (Schumberger)
- Duenweg (MPI Mainz - Germany)
- Messina (U. Duesselford - Germany)
- Warren (Unilever)
- Yeomans (U. Oxford - UK)
- Tao (U. Twente - Netherlands)
- Lowe (U. Amsterdam - Netherlands)

- Stratford (U. Edinburgh - UK)

Funding was provided by SoftComp, the European Union Network of Excellence.

http://www.tn.utwente.nl/cdr/WorkshopSoftcomp/index.html

Multiscale Modelling in Biological Systems

Organised by: Massimo Noro, Sophia Yaliraki, Greg Voth

The Conference

The conference is the first meeting of its kind and has the ambition to bring together the leading figures in the field of multiscale modelling, applied to biological problems. This event was sponsored by the Centre for Biophysical Modeling and Simulation (University of Utah), the Institute of Mathematical Sciences (Imperial College London), and Unilever.

The Model

This workshop was articulated around the Gordon Research Conference style with plenary talks in the morning and late afternoons, and plenty of time for formal and informal discussion.

The Scope

Presentations and discussions focussed around two themes:

- Deriving the Model: Which are acceptable equations/models at different scales?
- The Model at Work: What are the appropriate ways for obtaining input parameters (from a different scale) as they arise in biologically interesting systems where scales are inherently coupled.

Conference Program

The list of the conference program is the following:

- Gary Ayton, Center for Biophysical Modeling & Simulation, University of Utah, "Multiscale Simulation of Lipid Bilayers and Membrane Bound Proteins"
- Chakra Chennubhotla, Dept. of Computational Biology, Univ. of Pittsburgh, "Allosteric Communication in Supramolecular Systems: Application to Chaperonin GroEL/GroES"
- Wim Briels, Department of Science and Technology, University of Twente, The Netherlands. "Simulations of Polymers, Worms and Bilayers"
- Charles L. Brooks, Computational Biophysics & Chemistry, Scripps Research Institute "Multiscale Multiresolution Dynamics of Biological Assemblies Using Elastic Network Normal Mode Methods"
- Frank Brown, Department of Chemistry, University of California, Santa Barbara, "Elastic Models for Biomembrane Dynamics and Structure"
- Arup K. Chakraborty, Department of Chemical Engineering, Massachusetts Institute of Technolgy, "Intercellular Communication in the Adaptive Immune System: An Opportunity for Multiscale Models"
- Qiang Cui, Department of Chemistry, University of Wisconsin, Madison, "Development of Effective QM/MM and Coarse-grained Models for Multiscale Simulations of Biomolecules"
- Ron Elber, Computer Science, Cornell University, "Milestoning: An Approach to Coarse Graining Time along Complex Reaction Coordinates"
- Robert L. Jernigan, Laurence H. Baker Center for Bioinformatics and Biological Statistics, Iowa State University, "Coarser-graining and Mixed Coarse Graining of Structures"
- Richard Lavery, Laboratoire de Biochimie Thorique, Institut de Biologie Physico-Chimique. "Atomic-scale and Coarse-grain Approaches to Studying Protein Mechanics"
- Jianpeng Ma, Computational & Experimental Structural Biology & Cell Biology, Baylor College of Medicine, "A New Multiscale Monte Carlo Method for Biomolecular Simulation"
- Siewert-Jan Marrink, Department of Biophysical Chemistry, University of Groningen, The Netherlands, "Simulating Phase Transformations of Lipid Membranes"

- Massimo Noro, Unilever, "Modelling Stratum Corneum Lipids"
- Rob Phillips, Dept. of Applied Physics, Biochemistry & Molecular Biophysics, and Mechanical Engineering, California Institute of Technology, "The Multiscale Challenge of Crick's 'Two Great Polymer Languages' "
- John Straub, Department of Chemistry, Boston University. "Too Much or Not Enough? Coarse-grained Modeling of the Thermodynamics and Dynamics of Peptide Folding"
- Valentina Tozzini, NEST-INFM-CNR and Scuola Normale Superiore, Pisa, Italy "Extremely Coarse Grained Models for Large Time-scale Molecular Dynamics Simulations"
- Gregory A. Voth, Center for Biophysical Modeling & Simulation, University of Utah, "Systematic Coarse-graining and Multiscale Modeling of Biomolecular Systems"
- Sopia Yaliraki, Department of Chemistry, Imperial College, London. "Coarse-graining with Sum of Squares"

Upcoming Computational Physics Events

Conference on Computational Physics 2006

The Conference on Computational Physics (CCP) 2006 continues the series of the APS-EPS "Physics Computing". It takes place from August 29^{th} to September 1^{st}, 2006 at Gyeongju which is located 210 miles southeast of Seoul, the capital city of Korea.

Web page: http://ccp2006.postech.edu/

Advance announcement: The CCP 2007 will take place from 5 to 8 September 2007 in Brussels.

Conference on Computational Magnetism December 2006

The Computational Physics Group and the Magnetism Group of the Institute of Physics organise a one-day meeting on computational magnetism ranging from the atomic scale up to length-scales of micrometres. The relevant computational techniques focus around micromagnetism, Heisenberg models with dipolar interaction and ab-initio methods.

For each of these computational methods there will be an introductory talk providing an overview of the computational methods before leading researchers present their most recent methods and results.

There will be a poster session and plenty of opportunity to network.

Invited speakers include:

- Balasz Gyorffy (Bristol, UK), Ab-initio methods
- Bob McMichael (NIST, USA) Micromagnetism (OOMMF is developed and maintained at NIST)
- Uli Nowak (York, UK), Heisenberg models

The meeting takes place on Wednesday 13 December 2006 at the Institute of Physics (London).

Web page: http://conferences.iop.org/COMM/

Computational Tools

In this part of the newsletter, we provide occasionally information on selected software packages, tips and tricks relating to the Unix/Linux operating systems and other computational tools. (Contributions to the section are very welcome and can be emailed to Hans Fangohr.)

Visualisation

In this edition we provide pointers to two free (and open source) visualisation packages

- `Grace` – this is a "classic" program which does an excellent job on producing high-quality plots of the $y = f(x)$ type. It also supports barcharts, error bars and much more. It can be driven using a graphical user interface but can also be scripted using it's own command set. There are interfaces to other languages such as C and Python. (http://plasmagate.weizmann.ac.il/Grace/)
- `pylab` – if you are using Python for scientific work, then this is a very nice plotting package worth investigating. It can only be driven via Python commands but is ideally suited to automatically generate and save plots. (It is also known under its old name "matplotlib" because the syntax used to generate plots are very similar to Matlab. (http://matplotlib.sourceforge.net/)

Linux tools on Windows/Mac OS X

Linux/Unix is often used by computational scientists because it gives the user so much power and control, and allows to script all processes fairly easily.

- "Linux on Windows" – if you are running MS Windows and you would like your operating system to have Linux-like capabilities, checkout "CygWin". This provides all the core utilities of a Linux system for MS Windows. (http://www.cygwin.com/)
- "Linux on Mac OS X" – Mac OS X is based on BSD Unix. There are two management tools that provide easy access to all the standard Linux software. These are the "fink" project http://fink.sourceforge.net/ and "Darwinports" http://darwinports.opendarwin.org/.