Numerical Modelling in FORTRAN day 11

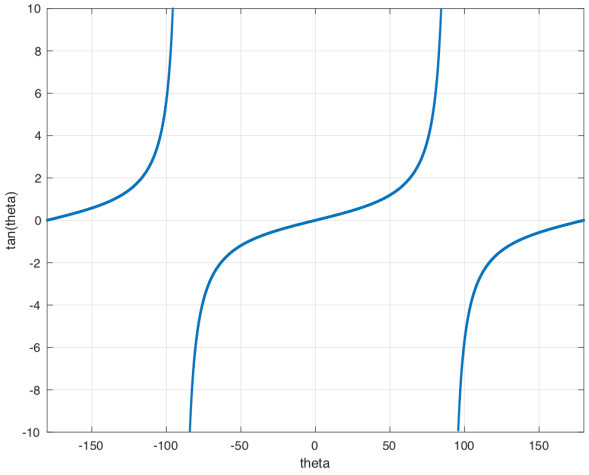
Paul Tackley, 2020

Today's Goals

1. **Fortran and numerical methods**: Review and discussion. New features of Fortran 2003, 2008 & 2018.
2. **Projects**: Define goals, identify information needed from me. Due date: 19 February 2021

Feeback on homework

- Use atan2(o,a) instead of atan(o/a) to get correct angle and avoid divide-by-0.
  - Each tan value maps to 2 possible angles in the range -180º to 180º
  - (-1,-1) has different angle to (1,1)
- Use built-in complex type
  - Access components using real() and aimag() or
  - (since f2008) c%re & c%im



Useful libraries and other software

- For standard operations such as matrix solution of systems of linear equations, taking fast-Fourier transforms, etc., it is convenient to download suitable routines from some free library, rather than writing your own from scratch.
- There are also more specialised free programs available for some applications
- A list is at: http://www.lahey.com/other.htm
- Much is available at: http://www.netlib.org/

Free Software http://www.lahey.com/other.htm

- BLAS, linear algebra subroutines
- DISLIN, plotting library compatible with LF95 for Windows or Linux
- DOS utility to unpack the shell archives available at Netlib, e.g., the BLAS routines above.
- EISPACK
- Elf90 example code
- f90ppr, a Fortran 90 preprocessor
- f90tops, a Fortran 90 to PostScript converter
- F90SPLIT, a Fortran 90 version of fsplit
- FLIB, a collection of Fortran routines with emphasis on non-numeric computations
- FLOPPY, a Fortran 77 coding convention checker and code tidier
- FLOW, produces reports based on the output of FLOPPY
- FTNCHEK, detects unused, undeclared, and uninitialized variables
- Fortran Library, a directory of user-submitted information of interest to scientific and technical programmers
- FXDR, subroutines to do unformatted I/O across platforms
- Harwell-Boeing collection(sparse matrices)
- LAPACK
- LINPACK
- Molecular dynamics Fortran 77 benchmark
- MINPACK
- MUDPACK
- Multiple Precision Arithmetic Package
- NASA, Open source Fortran code from NASA
- PSPLOT, a Fortran-callable, Postscript plotting library
- ODEPACK
- SLATEC
- Sparse Matrix Fortran 90 Library for LF90 by E. A. Meese
- SPECFUN
- STARPAC
- StopWatch, a Fortran 90 module for timing Fortran programs
- TOMS
- UNFSEQ.F90 - Program to convert unformatted sequential files from LF95 format to Lahey's LF90 and EM/32 format.

Some well-known libraries

- lapack95: common linear algebra problems: equations, least squares, eigenvalues etc.
  - http://www.netlib.org/lapack95/
  - also a parallel version scalapack http://www.netlib.org/scalapack/
- MPI: for running programs on multiple CPUs. 2 common versions are
  - MPICH: http://www.mpich.org/
  - Open MPI: http://www.open-mpi.org/
- PETSc: Portable, Extensible Toolkit for Scientific Computation: http://www.mcs.anl.gov/petsc/

Fortran Review

- History
  - 60 years since first compiler
  - versions 66, 77, 90, 95, 2003, 2008, 2018
- Variables
  - types: real, integer, logical, character, complex
  - Implicit types and implicit none
  - Initialisation
  - parameters
  - derived types
  - precision (32 vs 64 bit etc.)

Fortran Review (2)

- Arrays
  - fixed vs. automatic vs. allocatable vs. assumed shape
  - index ranges
  - data, reshape
  - built-in array algebra
- Input/Ouput
  - open, print, read, write, close
  - formats
  - iostat, rewind, status
  - namelist
  - binary vs. ascii, direct vs. sequential

Fortran Review (3)

- Flow control structures
  - do loops (simple, counting, while, concurrent), exit, cycle
  - if…elseif…endif blocks
  - select case…
  - forall
  - where

Fortran Review (4)

- Functions and subroutines (procedures)
  - Intrinsic functions: mathematical, conversion, …
  - internal (contains), external (& interface blocks), in modules
  - array functions
  - recursive functions and result statement
  - named arguments
  - optional arguments
  - generic procedures
  - overloading
  - user-defined operators
  - save

Fortran Review (5)

- Modules
  - use
  - public and private variables and procedures
  - =>
- Pointers
  - to variables, arrays, array sections, areas of memory, in defined types
- Optimization
  - maximize use of cache and pipelining
  - loops most important
  - 90/10 rule (focus on bottlenecks)
  - avoid branches, maximize data locality, unroll, tiling,…
- Libraries, makefiles

For a more detailed review & discussion, see:

- https://www.tacc.utexas.edu/documents/13601/162125/fortran_class.pdf

New features of Fortran 2003

- For a detailed description see: https://wg5-fortran.org/N1601-N1650/N1648.pdf
- For a summary: http://www.fortran.bcs.org/2007/jubilee/newfeatures.pdf
- Enhancements to derived types
  - Parameterized derived types
  - Improved control of accessibility
  - Improved structure constructors
  - Finalizers

Much of f2003's new functionality is related to Object-Oriented Programming

- From Wikipedia:
- "**Object-oriented programming** (**OOP**) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as *attributes* or *properties*), and code, in the form of procedures (often known as *methods*). A feature of objects is an object's procedures that can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another."

https://en.wikipedia.org/wiki/Object-oriented_programming
http://users.monash.edu/~cema/courses/CSE5910/lectureFiles/lecture1b.htm

OOP in Fortran 2003-

- Derived types are enhanced to include procedures (i.e. functions or subroutines)
  - Encapsulation – combining together all the data & procedures into one object (with extensive use of 'private' variables & procedures)
- Derived types can be extended to make new derived types (inheritance, hierarchy)
  - Class – a group of derived types that are related to each other
  - Polymorphism: variables and procedures can be any type within the same class, dynamic at run time

Useful introductions are at:

- http://fortranwiki.org/fortran/show/Object-oriented+programming
- http://annefou.github.io/Fortran/classes/classes.html
- https://gist.github.com/n-s-k/522f2669979ed6d0582b8e80cf6c95fd

New features of Fortran 2003 (2)

- Object-oriented programming support
  - Type extension and inheritance
  - Polymorphism
  - Dynamic type allocation
  - Type-bound procedures
- Data manipulation enhancements
  - Allocatable components
  - Deferred type parameters
  - VOLATILE attribute
  - Explicit type specification in array constructors & allocate statements
  - Extended initialization expressions
  - Enhanced intrinsic procedures

New features of Fortran 2003 (3)

- Input/output enhancements
  - Asynchronous transfer
  - Stream access
  - User specified transfer ops for derived types
  - User specified control of rounding
  - Named constants for preconnected units
  - FLUSH
  - Regularization of keywords
  - Access to error messages
- Procedure pointers

New features of Fortran 2003 (4)

- Support for the exceptions of the IEEE Floating Point Standard
- Interoperability with the C programming language
- Support for international characters (ISO 10646 4-byte characters...)
- Enhanced integration with host operating system
  - Access to command line arguments,environment variables, processor error messages
- Plus numerous minor enhancements

Fortran 2008

- Minor update of Fortran 2003
- Brief summary (Wikipedia)
  - Submodules – additional structuring facilities for modules; supersedes ISO/IEC TR 19767:2005
  - Coarray Fortran – a parallel execution model
  - The DO CONCURRENT construct – for loop iterations with no interdependencies
  - The CONTIGUOUS attribute – to specify storage layout restrictions
  - The BLOCK construct – can contain declarations of objects with construct scope
  - Recursive allocatable components – as an alternative to recursive pointers in derived types
- For more details see
  - http://fortranwiki.org/fortran/show/Fortran+2008
  - https://wg5-fortran.org/N1801-N1850/N1828.pdf

Fortran 2018

- A moderate update of Fortran 2008, with
  - Further interoperability of Fortran with C
  - Additional parallel features
  - Conformance to the latest floating-point standard
  - Various small features that address deficiencies and discrepancies
- For more information:
  - http://fortranwiki.org/fortran/show/Fortran+2018

get_environment_variable()

```
program testenv
  implicit none
  character(len=255) :: answer

  call get_environment_variable("HOME",answer)
  print*,"HOME=",trim(answer)

  call get_environment_variable("SHELL",answer)
  print*,"SHELL=",trim(answer)

  call get_environment_variable("TERM",answer)
  print*,"TERM=",trim(answer)

end program testenv
```

```
HOME=/Users/pjt
SHELL=/bin/bash
TERM=xterm-256color
```

get_environment_variable()

get_environment_variable (name,value,length,status,trim_name)

- name = name of environment variable (provided by user)
- value = the result (string)
- length (optional): the number of characters
- status (optional): 0=success, -1=value too short, 1=not exist, 2=not supported, >2=another error
- trim_name (optional): .true.= ignore trailing blanks when matching environment variable

Interoperability with C (f2003)

- New intrinsic module iso_c_binding
- Declare variables with (kind=c_float) etc.
- Need interface blocks for C procedures labelled bind(C)
- Derived types can be passed, if they are declared using type, bind(C)::
- Fortran procedures can be called from C if they are declared using bind(C)

Simple example: calling C from Fortran

```
program c_fortran

  use, intrinsic:: iso_c_binding
  implicit none

  integer (kind=c_int) :: ic
  real  (kind=c_float) :: r(0:99),a,b,c

  interface  ! interface for the C function
     subroutine do_something_in_c(a,b,c) bind(C)
       use, intrinsic :: iso_c_binding
       real (kind=c_float) :: a,b,c
     end subroutine do_something_in_c
  end interface

  print'(a,$)','Input two numbers:'
  read*,a,b
  call do_something_in_c (a,b,c)
  print*,'Hello from Fortran: a + b = ',c

end program c_fortran
```

The C routine

```
#include <stdio.h>

void do_something_in_c (float *a, float *b, float *c)
{
  *c = *a + *b;
  printf(" Hello from C: a + b = %f\n", *c);
}
```

```
Input two numbers:1 2
 Hello from C: a + b = 3.000000
 Hello from Fortran: a + b =    3.00000000
```

Table B-1:
Selected data types declared in module `iso_c_binding`

Fortran type	Fortran kind	C type
INTEGER	C_INT	int
	C_SHORT	short int
	C_LONG	long inrt
	C_LONG_LONG	long long int
	C_SIGNED_CHAR	signed char
		unsigned char
	C_SIZE_T	size_t
	C_INT8_T	int8_t
	C_INT16_T	int16_t
	C_INT32_T	int32_t
	C_INT64_T	int64_t
REAL	C_FLOAT	float
	C_DOUBLE	double
	C_LONG_DOUBLE	long double
COMPLEX	C_FLOAT_COMPLEX	float _Complex
	C_DOUBLE_COMPLEX	double _Complex
	C_LONG_DOUBLE_COMPLEX	long double _Complex
LOGICAL	C_BOOL	_Bool
CHARACTER	C_CHAR	char

From *Chapman, Fortran for Scientists and Engineers*

Table B-2:
Constants and procedures declared in module `iso_c_binding`

Name	Description
Named constants	
C_NULL_CHAR	Null character ('\0')
C_ALERT	Alert ('\a')
C_BACKSPACE	Backspace ('\b')
C_FORMFEED	Form feed ('\f')
C_NEW_LINE	New Line ('\n')
C_CARRIAGE_RETURN	Carriage return ('\r')
C_HORIZONTAL_TAB	Horizontal tab ('\t')
C_VERTICAL_TAB	Vertical tab ('\v')
Procedures	
C_ASSOCIATED	Function to test if a C pointer is associated.
C_F_POINTER	Function to convert a C pointer to a Fortran pointer.
C_F_PROCPOINTER	Function to convert a C function pointer to a Fortran procedure pointer.
C_FUNLOC	Returns the address in memory of a C function.
C_LOC	Returns the address in memory of a C data item.
C_SIZEOF	Returns the size of a C data item in bytes.
Types to interoperate with C pointers	
C_PTR	Derived type representing any C pointer type.
C_FUNPTR	Derived type representing any C function pointer type.
C_NULL_PTR	The value of a null C pointer.
C_NULL_FUNPTR	The value of a null C function pointer.

From *Chapman, Fortran for Scientists and Engineers*

Floating-point exception handling (f2003)

- Facilitates handling of floating-point exceptions in the IEEE standard. The 5 types are of exception:
  - Overflow, underflow, divide_by_zero, invalid, inexact
- Related to these are variable status:
  - Denormal, NaN (signalling or quiet), Inf, -Inf, -0
- Intrinsic module ieee_exceptions:
  - A set of procedures for setting & testing the flags & inquiring about the features.
- Intrinsic module ieee_arithmetic:
  - Provides IEEE arithmetic facilities.
- Intrinsic module ieee_features:
  - Contains named constants corresponding to the features.

Useful functions

ieee_is_nan() & ieee_is_finite()

```
program ieee

  use, intrinsic:: ieee_arithmetic
  implicit none
  real a,zero,one

  zero=0.
  one =1.
  a = zero/zero
  if (ieee_is_nan(a)) print*,'a is NaN'
  a = one/zero
  if (.not.ieee_is_finite(a)) print*,'a is infinite'

end program ieee
```

```
a is NaN
a is infinite
```

The oldest high-level programming language

Mother Tongues

Tracing the roots of computer languages through the ages

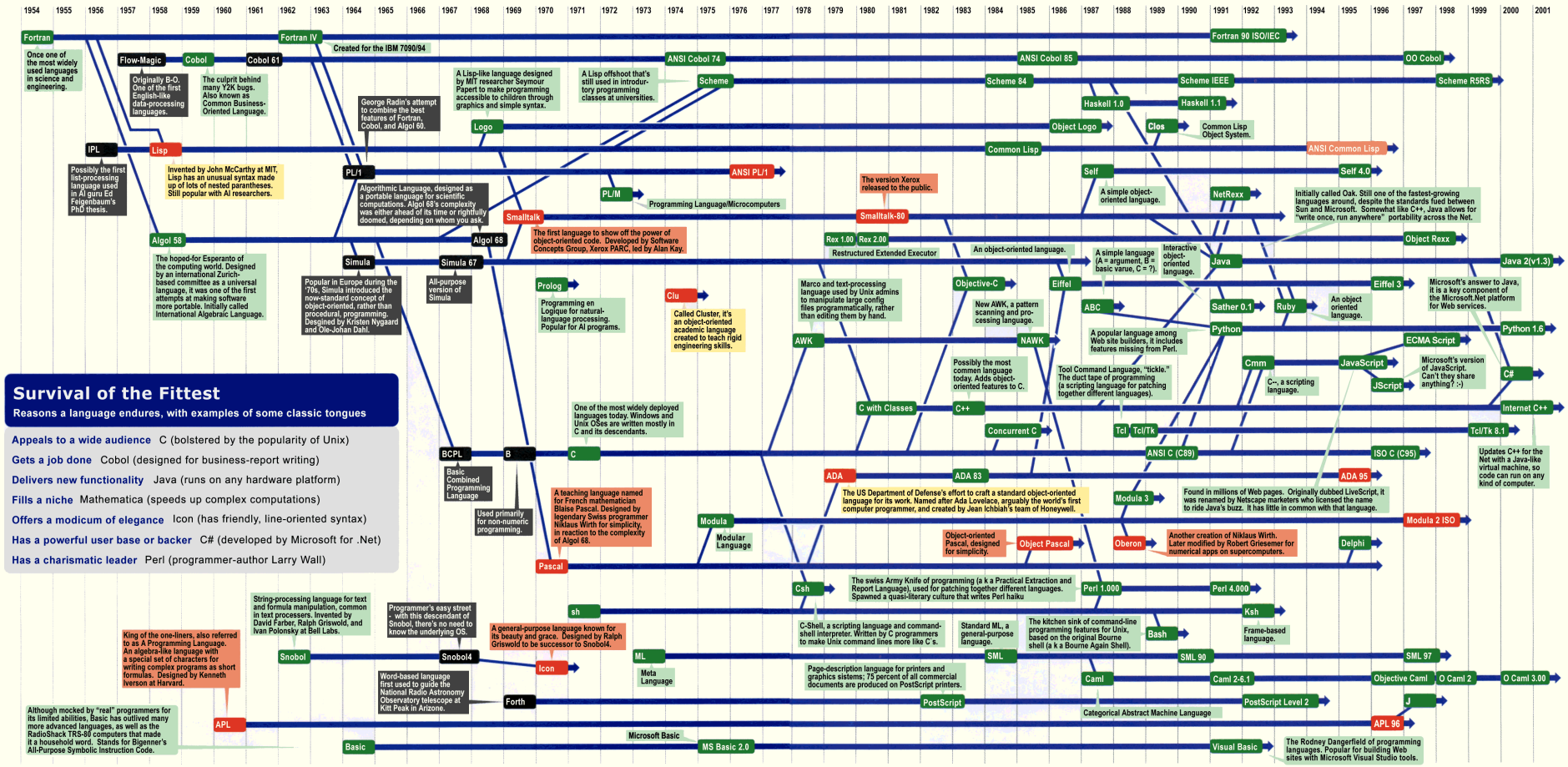
Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* - Michael Mendeno

Key

- 1954 Year Introduced
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**



Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Numerical methods review

- Approximations
  - Concept of discretization
  - Finite difference approximation
  - Treatment of boundary conditions
- Equations
  - diffusion equation
  - Poisson's equation
  - advection-diffusion equation
  - Stokes equation and Navier-Stokes equation
  - streamfunction and streamfunction-vorticity formulation
  - Pr, Ra, Ek
  - initial value vs. boundary value problems

Numerical methods Review (2)

- Timestepping
  - stability (timestep, advection scheme)
  - explicit vs. implicit, semi-implicit
  - upwind advection
- Solvers
  - direct
  - iterative (relaxation), multigrid method
  - Jacobi vs. Gauss-Seidel vs. red-black
- Parallelisation
  - Domain decomposition
  - Message-passing and MPI

Discussion

- We have focussed on one application (constant viscosity convection)
- Can apply same techniques to other applications / physical problems

Example: Darcy Equation
(groundwater or partial melt flow)

$$\vec{u} = -\frac{k}{\eta}\left(\vec{\nabla}P - g\rho\hat{\vec{y}}\right)$$

u=volume/area/time="Darcy velocity" (really a flux)

$$\vec{\nabla}\cdot\vec{u} = 0$$

k=permeability (related to porosity and interconnectivity)

η =viscosity of fluid, ρ =density of fluid, P=pressure in fluid

Simplify and rearrange

Take divergence: velocity is eliminated

$$\vec{\nabla}\cdot\vec{u} = -\vec{\nabla}\cdot\left(\frac{k}{\eta}\vec{\nabla}P\right) + \vec{\nabla}\cdot\left(\frac{k}{\eta}g\rho\hat{\vec{y}}\right) = 0$$

2nd order equation for pressure

$$\vec{\nabla}\cdot\left(\frac{k}{\eta}\vec{\nabla}P\right) = g\frac{\partial}{\partial y}\left(\frac{\rho k}{\eta}\right)$$

Combine k and η into hydraulic resistance R

$$\vec{\nabla}\cdot\left(\frac{1}{R}\vec{\nabla}P\right) = g\frac{\partial}{\partial y}\left(\frac{\rho}{R}\right)$$

A Poisson-like equation for pressure => easy to solve using existing methods

Example 2: Full elastic wave equation

Most convenient to solve as coupled first-order PDEs

F=ma for a continuum: $$\rho \frac{\partial \vec{v}}{\partial t} = \nabla \cdot \underline{\underline{\sigma}}$$

Hooke's law for an **isotropic** material $$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}$$

Where $$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$ u=displacement of point from equilibrium position

μ =shear modulus. λ, μ are known as Lamé constants

Sometimes written
Using K=bulk modulus

$$\sigma_{ij} = K\varepsilon_{kk}\delta_{ij} + 2\mu(\varepsilon_{ij} - \tfrac{1}{3}\varepsilon_{kk}\delta_{ij})$$

Resulting equations to solve using finite differences

Variables: 3 velocity components and 6 stress components

On staggered grid (same as viscous flow modelling)

$$\frac{\partial}{\partial t} v_x = \frac{1}{\rho}\left(\frac{\partial}{\partial x}\sigma_{xx} + \frac{\partial}{\partial y}\sigma_{xy} + \frac{\partial}{\partial z}\sigma_{xz}\right) \qquad \frac{\partial}{\partial t}\sigma_{xx} = (\lambda + 2\mu)\frac{\partial v_x}{\partial x} + \lambda\left(\frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}\right)$$

$$\frac{\partial}{\partial t} v_y = \frac{1}{\rho}\left(\frac{\partial}{\partial x}\sigma_{xy} + \frac{\partial}{\partial y}\sigma_{yy} + \frac{\partial}{\partial z}\sigma_{zy}\right) \qquad \frac{\partial}{\partial t}\sigma_{yy} = (\lambda + 2\mu)\frac{\partial v_y}{\partial y} + \lambda\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z}\right)$$

$$\frac{\partial}{\partial t} v_z = \frac{1}{\rho}\left(\frac{\partial}{\partial x}\sigma_{zx} + \frac{\partial}{\partial y}\sigma_{zy} + \frac{\partial}{\partial z}\sigma_{zz}\right) \qquad \frac{\partial}{\partial t}\sigma_{zz} = (\lambda + 2\mu)\frac{\partial v_z}{\partial z} + \lambda\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right)$$

$$\frac{\partial}{\partial t}\sigma_{xy} = \mu\left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}\right) \qquad \frac{\partial}{\partial t}\sigma_{xz} = \mu\left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x}\right) \qquad \frac{\partial}{\partial t}\sigma_{yz} = \mu\left(\frac{\partial v_y}{\partial z} + \frac{\partial v_z}{\partial y}\right)$$

Mathematical classification of 2nd-order PDEs: Elliptical, hyperbolic, parabolic

Elliptical $\nabla^2\phi = f$ e.g., Poisson

Parabolic $\frac{\partial\phi}{\partial t} \propto \nabla^2\phi$ e.g., diffusion

Hyperbolic $\nabla^2\phi \propto \frac{1}{V^2}\frac{\partial^2\phi}{\partial t^2}$ e.g., wave eqn.

We've done them all!

Waves: $$\frac{\partial^2 P}{\partial t^2} = v^2 \nabla^2 P$$

Fluids:

$$\frac{1}{\mathrm{Pr}}\left(\frac{\partial \vec{v}}{\partial t} + \vec{v}\cdot\vec{\nabla}\vec{v}\right) = -\vec{\nabla}P + \vec{\nabla}\cdot\left(\eta\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)\right) + \frac{1}{Ek}\vec{\Omega}\times\vec{v} + Ra.T\hat{g}$$

$$\frac{\partial T}{\partial t} + \vec{v}\cdot\nabla T = \nabla\cdot(k\nabla T) \qquad \nabla\cdot\vec{v} = 0$$

Percolation: $$\vec{\nabla}\cdot\left(\frac{k}{\eta}\vec{\nabla}P\right) = g\frac{\partial}{\partial y}\left(\frac{\rho k}{\eta}\right)$$

note the many similar terms!

N-body: $$\frac{d^2\vec{x}_i}{dt^2} = \vec{a}_i \qquad \vec{a}_i = G\sum_{j=1}^{n, j\neq i}\frac{m_j}{\left|\vec{x}_j - \vec{x}_i\right|^2}\cdot\frac{(\vec{x}_j - \vec{x}_i)}{\left|\vec{x}_j - \vec{x}_i\right|}$$

FORTRAN