Eilmer4: A tool for analysing hypersonic flows

Peter Jacobs, Rowan Gollan
as co-chief gardeners of the code, together with
many contributors, as listed later
School of Mechanical Engineering, University of Queensland

13-April-2017, 08-June-2017

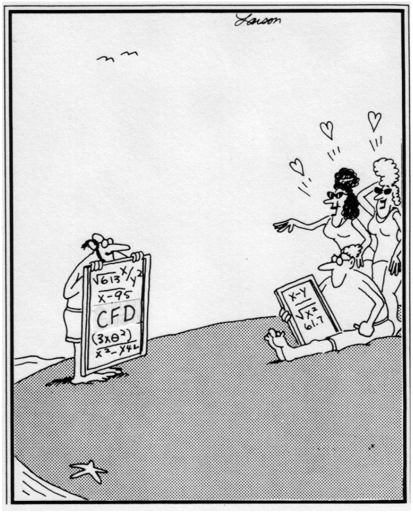
Motivation and History

Gas dynamic formulation and code implementation

Example – Powers and Aslam detonation wave

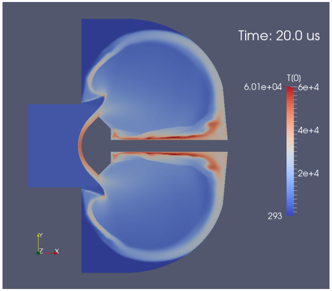
List of Contributors

Why build Computational Fluid Dynamic tools



- If we don't have suitable experience, we substitute (computational) analysis.
- As we develop new theories for hypersonic flows, we encode the models as computational tools.
- We want to be confident in our analytical tools...

Eilmer features – 1/2



- 2D/3D compressible flow simulation.
- Gas models include ideal, thermally perfect, equilibrium.
- Finite-rate chemistry.
- Inviscid, laminar, turbulent (k-ω) flow.
- Solid domains with conjugate heat transfer in 2D.
- User-controlled moving grid capability, with shock-fitting method for 2D geometries.
- Dense-gas thermodynamic models and rotating frames of reference for turbomachine modelling.

Eilmer features – 2/2

- Transient, time-accurate, using explicit Euler, PC, RK updates.
- Alternate steady-state solver with implicit updates using Newton-Krylov method.
- Parallel computation on a cluster computer, using MPI in Eilmer2,3 and shared memory in dgd/Eilmer4.
- Multiple block, structured and unstructured grids.
- Native grid generation and import capability.
- Unstructured-mesh partitioning via Metis.

- `en.wikipedia.org/wiki/Eilmer_of_Malmesbury`
- Gas model calculator and compressible flow relations.

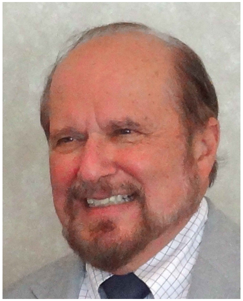
Origins

- in the late 1980s, the state of the art for scramjet simulations involving reactive flow was JP Drummond *SPARK* code
- Flow solver component based on Bob McCormack's (1969) finite-difference shock-capturing technique.
- All configuration hard-coded into the Fortran source code and compiled to run on a Cray supercomputer.
- In the 1980s, a new CFD technology (upwind flux) was being developed by the applied mathematics people and parallel computing environments were being developed by the computer science people (cluster computers).
- Dec 1990: following a CFD lesson on the chalk-board from Bob Walters and Bernard Grossman, *cns4u* was started with the intention to be like SPARK but with new technology

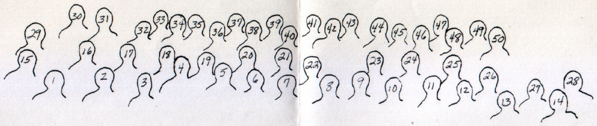
Origins – people

RW MacCormack

JP (Phil) Drummond


Bernard Grossman

Origins – people at ICASE in 1991



1. Stephen Otto
2. Leon Clancy
3. Peter Jacobs
4. M. Y. Hussaini
5. Gordon Erlebacher
6. Holly Joplin
7. Shelly Millen
8. Etta Blair
9. Barbara Cardasis
10. Linda Johnson
11. Emily Todd
12. Barbara Stewart
13. Rosa Milby
14. Cindy Cokus
15. Siva Thangam
16. Nicholas Blackaby
17. Ralph Smith
18. Sesh Venugopal
19. Yu Roung Ou
20. Thomas Crockett
21. John Otten
22. Shlomo Ta'asan
23. Mordechay Karpel
24. Rolf Radespiel
25. Kurt Bryan
26. Thomas Eidson
27. Peter Duck
28. Shahid Bokhari
29. Michael Arras
30. Peter Protzel
31. Luis Gomes
32. Saul Abarbanel
33. Scott Berryman
34. Fumio Kojima
35. Jeffrey Scroggs
36. Satyanarayan Gupta
37. Remi Abgrall
38. Eli Turkel
39. Philip Roe
40. Ravi Ponnusamy
41. James Quirk
42. Sutanu Sarkar
43. David Keyes
44. Daniel Joseph
45. Charles Speziale
46. Naomi Naik
47. John Van Rosendale
48. Piyush Mehrotra
49. Subhendu Das
50. Robert Voigt

Development of Eilmer

- 1993 built *sm3d*, a space-marching code for 3D scramjet flows
- 1995 through 1999: the postgrad years expanded scope of experimentation and application
- 1996: code reformulation around fluxes (frequent discussions with Mike Macrossan); all code still in C with a preprocessor having a little command interpreter built in.
- 1997: discovered scripting languages Tcl and Python
- May 2003: *scriptit.tcl* provided fully programmable environment for simulation-preparation.
- Aug 2004: *Elmer* began as a hybrid code using Python and C.
- Jun 2005: rewrite of *Elmer(2)* in C alone.
- Jul 2006: rewrite *Elmer2* in C++ and, in 2008, call it *Eilmer3*. Class-based implementation was easier to extend.

Eilmer4 – think big, but control the complexity.

- Jun 2015+: rebuild in the D and Lua programming languages.
- Heather Muir worked on the unstructured-grid generator. based on the paving algorithm.

Mathematical gas dynamics (in differential form)

Conservation of mass:

$$\frac{\partial}{\partial t}\rho + \nabla \cdot \rho \mathbf{u} = 0 \tag{1}$$

Conservation of species mass:

$$\frac{\partial}{\partial t}\rho_i + \nabla \cdot \rho_i \mathbf{u} = -\left(\nabla \cdot \mathbf{J}_i\right) + \dot{\omega}_i \tag{2}$$

Conservation of momentum:

$$\frac{\partial}{\partial t}\rho \mathbf{u} + \nabla \cdot \rho \mathbf{u}\mathbf{u} = -\nabla p - \nabla \cdot \left\{-\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\dagger}) + \tfrac{2}{3}\mu(\nabla \cdot \mathbf{u})\boldsymbol{\delta}\right\} \tag{3}$$

Conservation of total energy:

$$\begin{aligned}\frac{\partial}{\partial t}\rho E + \nabla \cdot (e + \frac{p}{\rho})\mathbf{u} = \nabla \cdot [k\nabla T + \sum_{s=1}^{N_v} k_{v,s}\nabla T_{v,s}] + \nabla \cdot \left[\sum_{i=1}^{N_s} h_i \mathbf{J}_i\right] \\ - \left(\nabla \cdot \left[\left\{-\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\dagger}) + \tfrac{2}{3}\mu(\nabla \cdot \mathbf{u})\boldsymbol{\delta}\right\} \cdot \mathbf{u}\right]\right) - Q_{\text{rad}}\end{aligned} \tag{4}$$

Conservation of vibrational energy:

$$\frac{\partial}{\partial t}\rho_i e_{v,i} + \nabla \cdot \rho_i e_{v,i}\mathbf{u} = \nabla \cdot [k_{v,i}\nabla T_{v,i}] - \nabla \cdot e_{v,i}\mathbf{J}_i + Q_{T-V_i} + Q_{V-V_i} + Q_{\text{Chem}-V_i} - Q_{\text{rad}_i} \tag{5}$$

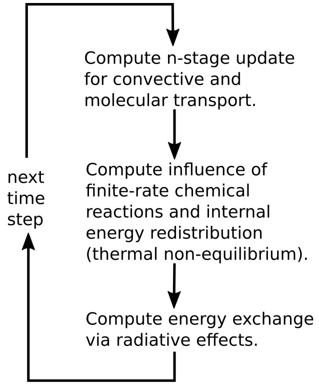
More maths...

Thermodynamic model of the gas...
Finite-rate chemical kinetics...
Radiation energy exchange...
Boundary conditions...
Features:

- 3D from the beginning, 2D as a special case
- structured- and unstructured-meshes for complex geometries
- refined thermochemistry
- moving meshes (Jason Qin and Kyle Damm)
- simplified and generalized boundary conditions
- coupled heat transfer
- shared-memory parallelism for multicore workstation use
- block-marching for speed (nenzfr and nozzle design)

Code structure

- D language data storage and solver, with embedded Lua interpreters for preprocessing, user-controlled run-time configuration in boundary conditions and source terms and thermochemical configuration.


Compute n-stage update for convective and molecular transport.
Compute influence of finite-rate chemical reactions and internal energy redistribution (thermal non-equilibrium).
Compute energy exchange via radiative effects.
next time step

```
for s=1 to n do:
    clear flux data
    apply pre-reconstruction action
    detect shock points
    reconstruct flow data at cell interfaces
    compute convective fluxes
    apply post-convective-flux action
    apply pre-spatial-derivative action
    compute spatial derivatves
    apply post-diffusion flux action
    add source terms, if any
    compute time derivatives of conserved quantities
    update cell-average conserved quantities for stage s
    decode conserved quantities to all flow quantities
```

New and Improved Thermochemistry

Compared to Eilmer3, the code is organised differently, into a couple of related classes.

- GasModel class
  - Describes the thermodynamic and molecular transport behaviour of the gas.
  - Connects pressure, temperature, internal energy and sound speed.
  - Estimates molecular-transport (diffusion) coefficients.
- ThermochemicalReactor class
  - Describes the finite-rate process to update (over a time step) the internal state of the gas.

GasModel class

Core functions in the gas model need to be provided by each specific gas model.

```
// Methods to be overridden.
//
abstract void update_thermo_from_pT(GasState Q);
abstract void update_thermo_from_rhoe(GasState Q);
abstract void update_thermo_from_rhoT(GasState Q);
abstract void update_thermo_from_rhop(GasState Q);
abstract void update_thermo_from_ps(GasState Q, double s);
abstract void update_thermo_from_hs(GasState Q, double h, double s);
abstract void update_sound_speed(GasState Q);
abstract void update_trans_coeffs(GasState Q);

abstract double dudT_const_v(in GasState Q);
abstract double dhdT_const_p(in GasState Q);
abstract double dpdrho_const_T(in GasState Q);
abstract double gas_constant(in GasState Q);
abstract double internal_energy(in GasState Q);
abstract double enthalpy(in GasState Q);
```

We have ideal gas, thermally-perfect mixture, ...

GasState class

A place to keep the data about the little bit of gas in a cell, or at other locations.

```
class GasState {
public:
    /// Thermodynamic properties.
    double rho;  /// density, kg/m**3
    double p;    /// presure, Pa
    double p_e;  /// electron pressure
    double a;    /// sound speed, m/s
    // For a gas in thermal equilibrium, all of the internal energies
    // are bundled together into u and are represented by a single
    // temperature Ttr.
    double Ttr;  /// thermal temperature, K
    double u;    /// specific thermal energy, J/kg
    // For a gas in thermal nonequilibrium, the internal energies are
    // stored unbundled, with u being the trans-rotational thermal energy.
    // The array length will be determined by the specific model and,
    // to get the total internal energy,
    // the gas-dynamics part of the code will need to sum the array elements.
    double[] e_modes;  /// specific internal energies for thermal nonequilibrium model, J/kg
    double[] T_modes;  /// temperatures for internal energies for thermal nonequilibrium, K
    /// Transport properties
    double mu;   /// viscosity, Pa.s
    double k;  /// thermal conductivity for a single temperature gas, W/(m.K)
    double[] k_modes;  /// thermal conductivities for the nonequilibrium model, W/(m.K)
    double sigma;    /// electrical conductivity, S/m
    /// Composition
    double[] massf;  /// species mass fractions
    double quality;  /// vapour quality
```

ThermochemicalReactor class

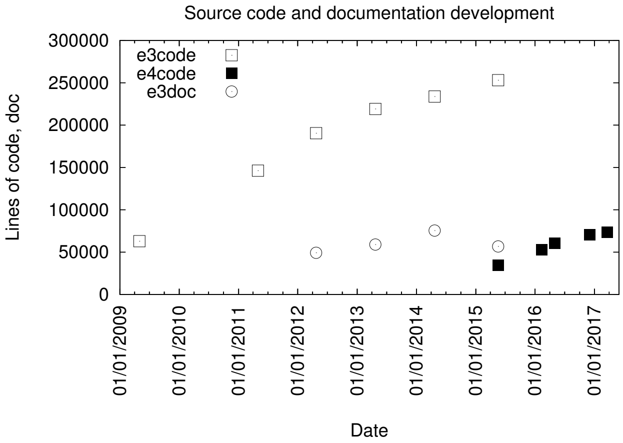
Like any good manager delegate the actual work to other code...

```
class ThermochemicalReactor {
public:
    this(GasModel gmodel)
    {
        // We need a reference to the original gas model object
        // to update the GasState data at a later time.
        _gmodel = gmodel;
    }

    // All the work happens when calling the concrete object
    // which updates the GasState over the (small) time, tInterval.
    abstract void opCall(GasState Q, double tInterval, ref double dtSuggest);

public:
    // We will need to access this referenced model from the Lua functions
    // so it needs to be public.
    GasModel _gmodel;
} // end class ThermochemicalReactor
```

Our work-horse code is a finite-rate reaction update using Arrhenius rate expressions, assuming a thermally-perfect gas mixture, however, we could use any other scheme.

How far have we gone, in lines of source code.

At 60 lines per page,
the Eilmer4 code is equivalent to a 1200 page document.



Verification and Validation Examples

Verification:

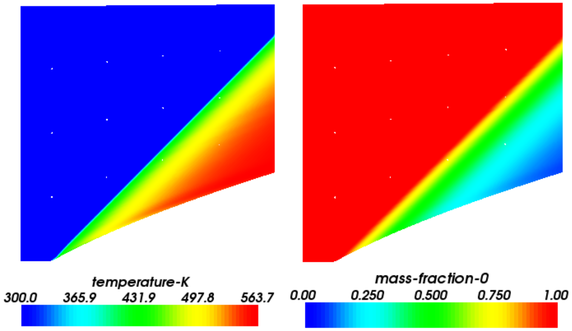
- Are we solving the equations correctly?
- Compare with numerical solutions from other codes.
- Some known exact (analytic) solution, possibly with limited physics.
- Manufactured solution that we must match (using special source terms and boundary conditions).

Validation:

- Are we solving the correct gas-dynamic equations?
- Compare with experimental measurements.

A detonation wave with simplified thermochemistry

The flow problem is an oblique detonation wave which is supported by a curved wedge surface. The analytical solution for this problem was first presented by Powers and Stewart (AIAA J. 1992).



In order to make the problem analytically tractable, the reaction mechanism for the detonation is simplified. The reaction is a one-step reaction that proceeds once an ignition temperature is reached.

The Powers and Aslam gas model

- Powers and Aslam (AIAA J 2006) used as a verification exercise.
- Two species A, B, with reaction of A to B proceeding at rate

$$\frac{d\rho_B}{dt} = \alpha\,\rho_A\,H(T - T_i)$$

  with rate constant $\alpha = 0.001\,\mathrm{s}^{-1}$
- Reaction progress variable is mass fraction of B: $\lambda = Y_B = \frac{\rho_B}{\rho}$
- $Y_A \;=\; 1 - Y_B$
- Equation of state for internal energy:

$$u = \frac{1}{\gamma - 1}\,\frac{p}{\rho} \;-\; \lambda q \;=\; C_v T \;-\; Y_B q$$

  with heat of reaction $q = 300000\,\mathrm{J/kg}$ and ratio of specific heats $\gamma = 6/5$.
- Pressure: $p \;=\; \rho R T$, with gas constant $R = 287\,\mathrm{J/kg.K}$

Powers and Aslam gas model functions

Code for the thermodynamic functions that compute the gas state. These functions override the functions (with corresponding names) in the GasModel base class.

```
override void update_thermo_from_pT(GasState Q) const
{
    Q.rho = Q.p/(Q.Ttr*_Rgas);
    Q.u = _Cv*Q.Ttr - Q.massf[1]*_q;
}
override void update_thermo_from_rhoe(GasState Q) const
{
    Q.Ttr = (Q.u + Q.massf[1]*_q)/_Cv;
    Q.p = Q.rho*_Rgas*Q.Ttr;
}
override void update_thermo_from_rhoT(GasState Q) const
{
    Q.p = Q.rho*_Rgas*Q.Ttr;
    Q.u = _Cv*Q.Ttr - Q.massf[1]*_q;
}
override void update_thermo_from_rhop(GasState Q) const
{
    Q.Ttr = Q.p/(Q.rho*_Rgas);
    Q.u = _Cv*Q.Ttr - Q.massf[1]*_q;
}
```

Powers and Aslam thermochemical reactor update

```
override void opCall(GasState Q, double tInterval, ref double dtSuggest)
{
    if (Q.Ttr > _Ti) {
        // We are above the ignition point, proceed with reaction.
        double massfA = Q.massf[0];
        double massfB = Q.massf[1];
        // This gas has a very simple reaction scheme that can be integrated    ↲
        explicitly.
        massfA = massfA*exp(-_alpha*tInterval);
        massfB = 1.0 - massfA;
        Q.massf[0] = massfA; Q.massf[1] = massfB;
    } else {
        // do nothing, since we are below the ignition temperature
    }
    // Since the internal energy and density in the (isolated) reactor is fixed,
    // we need to evaluate the new temperature, pressure, etc.
    _gmodel.update_thermo_from_rhoe(Q);
    _gmodel.update_sound_speed(Q);
}
```

Thermochemical input file

```
model = "PowersAslamGas"

PowersAslamGas = {
  R = 287,
  gamma = 6/5,
  q = 300000,
  alpha = 1000,
  Ti = 362.58
}
```

Input script – gas model and flow

```lua
config.title = "Oblique detonation wave with Powers-Aslam gas model."
print(config.title)
config.dimensions = 2

nsp, nmodes, gm = setGasModel('powers-aslam-gas-model.lua')
print("GasModel has nsp= ", nsp, " nmodes= ", nmodes)
massf1 = {A=1.0, B=0.0}
initial = FlowState:new{p=28.7e3, T=300.0, massf=massf1}
inflow = FlowState:new{p=86.1e3, T=300.0, velx=964.302, massf=massf1}
```

Notes:

- user's input script is Lua source code
- arguments to function calls delimited by ()
- tables delimited by {}
- object model by convention as described in Ierusalimschy's book "Programming in Lua"

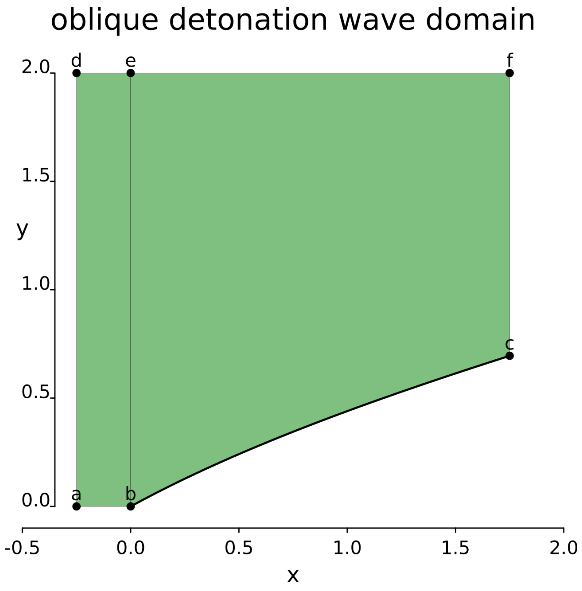
Input script – geometry definition: parameters and points

```lua
-- Geometry
xmin = -0.25
xmax = 1.75
ymin = 0.0
ymax = 2.0

dofile("analytic.lua")
myWallFn = create_wall_function(0.0, xmax)

-- Set up two patches in the (x,y)-plane by first defining
-- the corner nodes, then the lines between those corners.
a = Vector3:new{x=xmin, y=0.0}
b = Vector3:new{x=0.0, y=0.0}
c = Vector3:new{x=myWallFn(1.0).x, y=myWallFn(1.0).y}
d = Vector3:new{x=xmin, y=ymax}
e = Vector3:new{x=0.0, y=ymax}
f = Vector3:new{x=xmax, y=ymax}
```

Notes:

- Make your life simple; use good symbolic names.
- The full Lua interpreter is available to build complex functions.

Oblique detonation wave – geometry



Input script – geometry definition: lines, patches, grids

```
south0 = Line:new{p0=a, p1=b} -- upstream of wedge
south1 = LuaFnPath:new{luaFnName="myWallFn"} -- wedge surface
north0 = Line:new{p0=d, p1=e}; north1 = Line:new{p0=e, p1=f}
west0 = Line:new{p0=a, p1=d} -- inflow boundary
east0west1 = Line:new{p0=b, p1=e} -- vertical line, between patches
east1 = Line:new{p0=c, p1=f} -- outflow boundary
patch0 = makePatch{north=north0, east=east0west1, south=south0, west=west0}
patch1 = makePatch{north=north1, east=east1, south=south1, west=east0west1}
-- Mesh the patches, with particular discretisation.
factor = 2 -- for adjusting the grid resolution
nxcells = math.floor(40*factor)
nycells = math.floor(40*factor)
fraction0 = (0-xmin)/(xmax-xmin) -- fraction of domain upstream of wedge
nx0 = math.floor(fraction0*nxcells); nx1 = nxcells-nx0; ny = nycells
grid0 = StructuredGrid:new{psurface=patch0, niv=nx0+1, njv=ny+1}
grid1 = StructuredGrid:new{psurface=patch1, niv=nx1+1, njv=ny+1}
```

Notes:

- Fully-parametric grid generator is available.
- Table entries are mostly named. This is an advantage for large numbers of parameters and helps to make your input script self-documenting.

Input script – flow-domain with boundary conditions

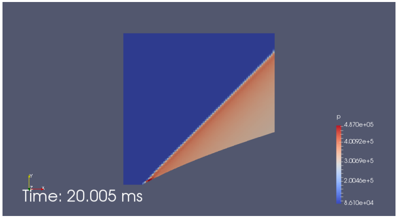
```
-- Define the flow-solution blocks and set boundary conditions.
-- We split the patches into roughly equal blocks so that
-- we make good use of our multicore machines.
blk0 = SBlockArray{grid=grid0, fillCondition=inflow, nib=1, njb=2,
                   bcList={west=InFlowBC_Supersonic:new{flowCondition=inflow},
                           north=OutFlowBC_Simple:new{}}}
blk1 = SBlockArray{grid=grid1, fillCondition=initial, nib=7, njb=2,
                   bcList={east=OutFlowBC_Simple:new{},
                           north=OutFlowBC_Simple:new{}}}
identifyBlockConnections()
```

Notes:

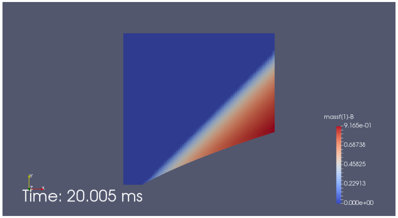
- May define many blocks on a single grid.
- We attach boundary conditions to the domain and specify the initial flow condition.
- Boundary conditions default to class WallBC_WithSlip.
- Some boundary conditions need extra information.

Oblique detonation wave – steady flow field

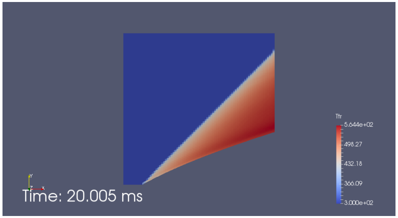
Pressure [86.1kPa – 487kPa]



Mass fraction B [0 – 0.9165]



Temperature [300K – 564.4K]



Shock wave angle $\beta \approx 45.2^\circ$

Run time is 2 minutes

Source code and documentation

Online source repositories are public:

- The *CF*CFD project
  `http://cfcfd.mechmining.uq.edu.au/`
- Eilmer4 code and documentation
  `https://bitbucket.org/cfcfd/dgd`
- To get started, clone the bitbucket repository.

Documentation in the form of user's guides:

- Unsteady flow solver.
- Geometry and grid builder.
- Simple thermochemical models.

Examples in Eilmer4 and Eilmer3 repositories are a good source of modelling ideas.

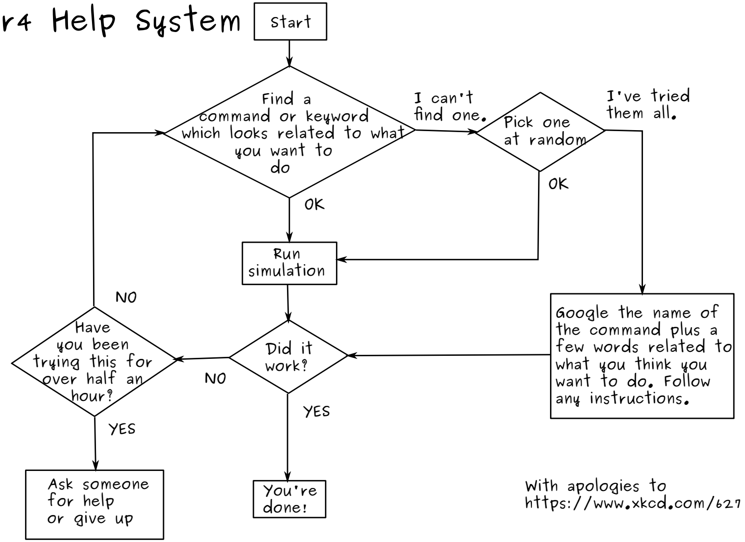
The Many Contributors...

Ghassan Al'Doori, Nikhil Banerji, Justin Beri, Peter Blyton, Daryl Bond, Arianna Bosco, Djamel Boutamine, Laurie Brown, James Burgess, David Buttsworth, Wilson Chan, Sam Chiu, Chris Craddock, Brian Cook, Jason Czapla, Kyle Damm, Andrew Dann, Andrew Denman, Zac Denman, Luke Doherty, Elise Fahy, Antonia Flocco, Delphine Francois, James Fuata, Nick Gibbons, David Gildfind, Richard Goozeé, Sangdi Gu, Stefan Hess, Jonathan Ho, Jimmy-John Hoste, Carolyn Jacobs, Ingo Jahn, Chris James, Ian Johnston, Ojas Joshi, Xin Kang, Rainer Kirchhartz, Sam Lamboo, Cor Lerink, Steven Lewis, Pierre Mariotto, Tom Marty, Matt McGilvray, David Mee, Carlos de Miranda-Ventura, Luke Montgomery, Heather Muir, Jan-Pieter Nap, Brendan O'Flaherty, Andrew Pastrello, Paul Petrie-Repar, Jorge Sancho Ponce, Daniel Potter, Jason (Kan) Qin, Deepak Ramanath, Andrew Rowlands, Michael Scott, Umar Sheikh, Sam Stennett, Ben Stewart, Joseph Tang, Katsu Tanimizu, Augustin Tibere-Inglesse, Pierpaolo Toniato, Paul van der Laan, Tjarke van Jindelt, Anand Veeraragavan, Jaidev Vesudevan, Jiangyong Wang, Han Wei, Mike Wendt, Brad (The Beast) Wheatley, Vince Wheatley, Lachlan Whyborn, Adriaan Window, Hannes Wojciak, Fabian Zander, Mengmeng Zhao

...who have helped us...

collect the Low-hanging Fruits of Hypersonics.

Eilmer4 Help System


Start
Find a command or keyword which looks related to what you want to do
I can't find one.
Pick one at random
I've tried them all.
OK
OK
Run simulation
Google the name of the command plus a few words related to what you think you want to do. Follow any instructions.
Did it work?
NO
YES
Have you been trying this for over half an hour?
NO
YES
Ask someone for help or give up
You're done!

With apologies to https://www.xkcd.com/627