How to write good code for physical problems

or

Trying is the first step towards failure.

1. The Casimir Effect

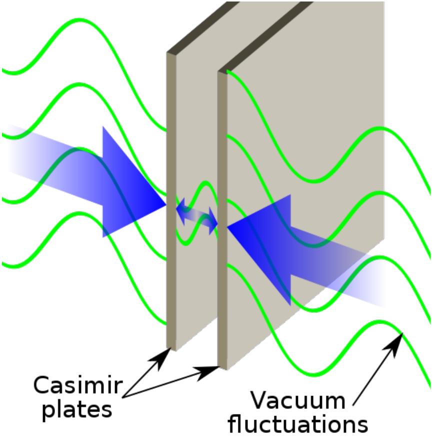
2. Unit tests

3. Revision control systems

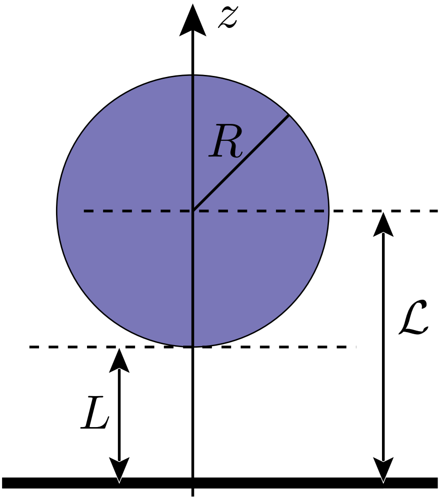
4. Optimization

5. Bits and pieces
   - Reproducability
   - Style
   - C and gcc

The Casimir Effect



Geometry plane–sphere



Formulas...

- Free Energy:

$$\mathcal{F} = 2k_{\mathrm{B}}T \sum_{n=0}^{\infty}{}' \sum_{m=0}^{\infty}{}' \log\det\left[\mathbb{1} - \mathcal{M}^{(m)}(\xi_n)\right]$$

- Matsubara-Frequencies:

$$\xi_n = \frac{2\pi n k_{\mathrm{B}} T}{\hbar}$$

- Round-Trip–Operator:

$$\mathcal{M}^{(m)}(\xi_n) = \begin{pmatrix} \mathcal{M}^{(m)}(E,E) & \mathcal{M}^{(m)}(E,M) \\ \mathcal{M}^{(m)}(M,E) & \mathcal{M}^{(m)}(M,M) \end{pmatrix}$$

and even more formulas...

- Matrix elements:

$$\mathcal{M}^{(m)}(E,E)_{\ell_1\ell_2} = \Lambda^{(m)}_{\ell_1\ell_2} a_{\ell_1} \left[A^{(m)}_{\ell_1\ell_2,\mathrm{TE}} + B^{(m)}_{\ell_1\ell_2,\mathrm{TM}}\right]$$
$$\mathcal{M}^{(m)}(M,M)_{\ell_1\ell_2} = \Lambda^{(m)}_{\ell_1\ell_2} b_{\ell_1} \left[A^{(m)}_{\ell_1\ell_2,\mathrm{TM}} + B^{(m)}_{\ell_1\ell_2,\mathrm{TE}}\right]$$
$$\mathcal{M}^{(m)}(E,M)_{\ell_1\ell_2} = \Lambda^{(m)}_{\ell_1\ell_2} a_{\ell_1} \left[C^{(m)}_{\ell_1\ell_2,\mathrm{TE}} + D^{(m)}_{\ell_1\ell_2,\mathrm{TM}}\right]$$
$$\mathcal{M}^{(m)}(M,E)_{\ell_1\ell_2} = -\Lambda^{(m)}_{\ell_1\ell_2} b_{\ell_1} \left[C^{(m)}_{\ell_1\ell_2,\mathrm{TM}} + D^{(m)}_{\ell_1\ell_2,\mathrm{TE}}\right]$$

- a_ℓ, b_ℓ: Mie-coefficients
- $\Lambda_{\ell_1\ell_2}$ prefactor

$$\Lambda^{(m)}_{\ell_1\ell_2} = -\sqrt{\frac{(2\ell_1+1)\,(2\ell_2+1)\,(\ell_1-m)!\,(\ell_2-m)!}{(\ell_1+m)!\,(\ell_2+m)!\,\ell_1(\ell_1+1)\,\ell_2(\ell_2+1)}}$$

Starting situation

The starting situation:

- you did the derivation
- you have a bunch of probably complicated formulas
- you can't solve the problem analytically
- you need code that solves your problem

From here:

- top to bottom
- bottom to top

Starting situation

The starting situation:

- you did the derivation
- you have a bunch of probably complicated formulas
- you can't solve the problem analytically
- you need code that solves your problem

From here:

- top to bottom
- bottom to top

Let's start bottom to top!

Let's start with the prefactor!

$$\Lambda_{\ell_1\ell_2}^{(m)} = -\sqrt{\frac{(2\ell_1+1)\,(2\ell_2+1)\,(\ell_1-m)!\,(\ell_2-m)!}{(\ell_1+m)!\,(\ell_2+m)!\,\ell_1(\ell_1+1)\,\ell_2(\ell_2+1)}}$$

The code:

```
from __future__ import division
from math import sqrt, factorial as fac

def Lambda(l1,l2,m):
    num   = (2*l1+1)*(2*l2+1)*fac(l1-m)*fac(l2-m)
    denom = fac(l1+m)*fac(l2+m)*l1*(l1+1)*l2*(l2+1)
    return -sqrt(num/denom)
```

Testing

So, what now?

Unit-Tests

Idea: At least one test for every function you write

benefits:

- find problems early
- avoid regressions
- documentation

It's easy! There are modueles for almost every language!

The test

```
from __future__ import division
from casimir import *
import unittest

class CasimirTest(unittest.TestCase):
  def test_lambda(self):
    self.assertAlmostEqual(Lambda(1,1,0)/(-1.5), 1)
    self.assertAlmostEqual(Lambda(20,15,4)/(-1.18121789e-11), 1)
    self.assertAlmostEqual(Lambda(80,80,80)/(-5.26980602e-287), 1)

if __name__ == "__main__":
    unittest.main()
```

Output

```
F
======================================================================
FAIL: test_lambda (__main__.CasimirTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "casimir_test.py", line 9, in test_lambda
    self.assertAlmostEqual(Lambda(80,80,80)/(-5.26980602e-287), 1)
AssertionError: 0.0 != 1 within 7 places

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

What went wrong?

$$\Lambda^{(m)}_{\ell_1\ell_2} = -\sqrt{\frac{(2\ell_1+1)\,(2\ell_2+1)\,(\ell_1-m)!\,(\ell_2-m)!}{(\ell_1+m)!\,(\ell_2+m)!\,\ell_1(\ell_1+1)\,\ell_2(\ell_2+1)}}$$

- $n!$ becomes large
- numerator or denumerator may extend range of doubles
- `denom` $\approx 9.3310^{576}$
- division cannot take place

Solution: Avoid division

Solution

Use logarithms!

$$
\begin{aligned}
\Lambda_{\ell_1 \ell_2}^{(m)} &= -\sqrt{\frac{(2\ell_1+1)\,(2\ell_2+1)\,(\ell_1-m)!\,(\ell_2-m)!}{(\ell_1+m)!\,(\ell_2+m)!\,\ell_1(\ell_1+1)\,\ell_2(\ell_2+1)}} \\
&= \sqrt{\frac{(2\ell_1+1)\,(2\ell_2+1)}{\ell_1(\ell_1+1)\,\ell_2(\ell_2+1)}} \\
&\times \exp\left[\frac{\log(\ell_1-m)! - \log(\ell_1+m)! + \ldots}{2}\right]
\end{aligned}
$$

The second try

Code:

```
def Lambda(l1,l2,m):
    return -sqrt( (2*l1+1)*(2*l2+1) / \
                  (l1*l2*(l1+1)*(l2+1)) ) \
      * exp( (lgamma(l1-m+1)+lgamma(l2-m+1)\
             -lgamma(l1+m+1)-lgamma(l2+m+1))/2 )

```

The second try

Code:

```
def Lambda(l1,l2,m):
    return -sqrt( (2*l1+1)*(2*l2+1) / \
                  (l1*l2*(l1+1)*(l2+1)) ) \
      * exp( (lgamma(l1-m+1)+lgamma(l2-m+1)\
             -lgamma(l1+m+1)-lgamma(l2+m+1))/2 )
```

And the test works!

Let's pause for a moment!

- numerical code can be tricky
- …especially floating point arithmetics
- unit tests can help finding bugs!
- unit tests can help preventing bugs!

- you can test a function…
- or your whole program

The test

```
from Casimir import PerfectReflectors
import unittest

class CasimirTest(unittest.TestCase):
  def test_casimir(self):
    ScriptL = 2e-6
    R = 1e-6
    T = 50
    Fexp = 2.95192899663732e-22
    lmax = 10
    nmax = 100
    casimir = PerfectReflectors(R,ScriptL,T,lmax=lmax)
    Fcalc = casimir.F(nmax=nmax)
    self.assertAlmostEqual(Fexp/Fcalc, 1)

if __name__ == "__main__":
    unittest.main()
```

Revision control systems

- example: Wikipedia editions
- you can commit changes
- you can collaborate
- you can see the differences between versions
- you can see the difference between the last (commited) version and your current code
- it is also kind of a backup
- it will document your work
- it will save you time!
- examples: git or subversion

Example

- you spend hours rewriting your code
- now it doesn't work anymore
- use a diff

```
@@ -632,13 +638,16 @@ double casimir_logdetD(casimir_t *self, int n, int m, casimir_mie_cache_t *cache
            gsl_matrix_set(M, dim+l2-min, dim+l1-min, pow(-1, l1+l2)*bl2*(cint.A_TM+cint.B_TE)); /* M_MM */

            gsl_matrix_set(M, dim+l1-min,     l2-min, al1*(cint.C_TE+cint.D_TM)); /* M_EM */
-           gsl_matrix_set(M, dim+l2-min,     l1-min, pow(-1, l1+l2+1)*al2*(cint.D_TE+cint.C_TM)); /* M_EM */
+           gsl_matrix_set(M, dim+l2-min,     l1-min, pow(-1, l1+l1+1)*al2*(cint.D_TE+cint.C_TM)); /* M_EM */

            gsl_matrix_set(M,     l1-min, dim+l2-min, bl1*(cint.C_TM+cint.D_TE)); /* - M_ME */
            gsl_matrix_set(M,     l2-min, dim+l1-min, pow(-1, l1+l2+1)*bl2*(cint.D_TM+cint.C_TE)); /* - M_ME */
```

My code is running too slow? What can I do?

Optimization

- don't optimize at an early stage!
- use a profiler and find why your program is slow
- use a better algorithm
- do you have to calculate everything?
- do you calculate things to often?
- exploit symmetries
- use caches
- if C: use optimization, inlines and macros

Example

What about A, B, C and D?

$$A^{(m)}_{\ell_1\ell_2,p}(\xi) = \frac{m^2\xi}{\mathrm{c}} \int_0^\infty \mathrm{d}k \, \frac{1}{k\kappa} \, r_p \, \mathrm{e}^{-2\kappa\mathcal{L}} \, \mathrm{P}^m_{\ell_1}\left(\frac{\kappa \mathrm{c}}{\xi}\right) \mathrm{P}^m_{\ell_2}\left(-\frac{\kappa \mathrm{c}}{\xi}\right)$$

$$B^{(m)}_{\ell_1\ell_2,p}(\xi) = \frac{\mathrm{c}^3}{\xi^3} \int_0^\infty \mathrm{d}k \, \frac{k^3}{\kappa} \, r_p \, \mathrm{e}^{-2\kappa\mathcal{L}} \, \mathrm{P}^{m\prime}_{\ell_1}\left(\frac{\kappa \mathrm{c}}{\xi}\right) \mathrm{P}^{m\prime}_{\ell_2}\left(-\frac{\kappa \mathrm{c}}{\xi}\right)$$

$$C^{(m)}_{\ell_1\ell_2,p}(\xi) = -\frac{im\mathrm{c}}{\xi} \int_0^\infty \mathrm{d}k \, \frac{k}{\kappa} \, r_p \, \mathrm{e}^{-2\kappa\mathcal{L}} \, \mathrm{P}^m_{\ell_1}\left(\frac{\kappa \mathrm{c}}{\xi}\right) \mathrm{P}^{m\prime}_{\ell_2}\left(-\frac{\kappa \mathrm{c}}{\xi}\right)$$

$$D^{(m)}_{\ell_1\ell_2,p}(\xi) = -\frac{im\mathrm{c}}{\xi} \int_0^\infty \mathrm{d}k \, \frac{k}{\kappa} \, r_p \, \mathrm{e}^{-2\kappa\mathcal{L}} \, \mathrm{P}^{m\prime}_{\ell_1}\left(\frac{\kappa \mathrm{c}}{\xi}\right) \mathrm{P}^m_{\ell_2}\left(-\frac{\kappa \mathrm{c}}{\xi}\right)$$

$$\kappa = \sqrt{\frac{\xi^2}{\mathrm{c}^2} + k^2}$$

Example

- my first approach: Use integration of scipy module
- my first result: Horrible slow, integration often doesn't converge
- solution: investigate properties of integral

Example

After substituation:

$$A^{(m)}_{\ell_1\ell_2} = A_0 \int_0^\infty \mathrm{d}x \, \frac{\mathrm{e}^{-x}}{x^2 + 2\tilde{\xi}x} \mathrm{P}^m_{\ell_1}\left(1 + \frac{x}{\tilde{\xi}}\right) \mathrm{P}^m_{\ell_2}\left(1 + \frac{x}{\tilde{\xi}}\right)$$

$$B^{(m)}_{\ell_1\ell_2} = B_0 \int_0^\infty \mathrm{d}x \, \left(x^2 + 2\tilde{\xi}x\right) \mathrm{e}^{-x} \mathrm{P}^{m\prime}_{\ell_1}\left(1 + \frac{x}{\tilde{\xi}}\right) \mathrm{P}^{m\prime}_{\ell_2}\left(1 + \frac{x}{\tilde{\xi}}\right)$$

$$C^{(m)}_{\ell_1\ell_2} = C_0 \int_0^\infty \mathrm{d}x \, \mathrm{e}^{-x} \, \mathrm{P}^m_{\ell_1}\left(1 + \frac{x}{\tilde{\xi}}\right) \mathrm{P}^{m\prime}_{\ell_2}\left(1 + \frac{x}{\tilde{\xi}}\right)$$

$$D^{(m)}_{\ell_1\ell_2} = D_0 \int_0^\infty \mathrm{d}x \, \mathrm{e}^{-x} \, \mathrm{P}^{m\prime}_{\ell_1}\left(1 + \frac{x}{\tilde{\xi}}\right) \mathrm{P}^m_{\ell_2}\left(1 + \frac{x}{\tilde{\xi}}\right)$$

$$\tilde{\xi} = 2\mathcal{L}\frac{\xi}{\mathrm{c}}$$

Example

- integrands are of the form $f(x)e^{-x}$, where $f(x)$ is a polynomial
- use Gauss-Laguerre: $\int_0^\infty \mathrm{e}^{-x} f(x) \approx \sum_i w_i f(x_i)$
- error $\propto f^{(2n)}(x)$
- if A, B, C, D are calculated as a vector: one must only compute associated legendre polynomial once
- there are symmetries between A and B, C and D
- Better algorithm:
  program runs way faster, results are exact (within double precision)

Example II

- Free Energy:

$$\mathcal{F} = 2k_{\mathrm{B}}T \sum_{n=0}^{n_{\max}}{}' \sum_{m=0}^{l_{\max}}{}' \log\det\left[\mathbb{1} - \mathcal{M}^{(m)}(\xi_n)\right]$$

- summands may be small when m increases
- one may crop summation over m

Profiler: gprof

- compile with flag `-pg`
- run program
- gprof program

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 67.80      1.64     1.64  5801418     0.00     0.00  _plm_array
 14.88      2.00     0.36  5801418     0.00     0.00  plm_Yl12md
 12.82      2.31     0.31  5801418     0.00     0.00  casimir_integrands_vec
  2.48      2.37     0.06   254646     0.00     0.01  gausslaguerre_integrate_vec
  0.83      2.39     0.02  5801418     0.00     0.00  casimir_rTM
  0.83      2.41     0.02      817     0.02     2.95  casimir_logdetD
  0.41      2.42     0.01                             plm_dPlm
  0.00      2.42     0.00   254646     0.00     0.01  casimir_integrate
  0.00      2.42     0.00    27349     0.00     0.00  casimir_Xi
  0.00      2.42     0.00      878     0.00     0.00  casimir_logdet1m
  0.00      2.42     0.00      878     0.00     0.00  la_norm_froebenius
  0.00      2.42     0.00      878     0.00     0.00  logdet1m_eigenvalues
  0.00      2.42     0.00       19     0.00     0.00  casimir_mie_cache_alloc
```

Reproducability

- what were the parameters given?
- what version of the code were you using?
- what changes?
- when?
- on which machine?
- append data to plots

Watch your style!

- Use indention
- Use meaningful names for your variables
- Use comments
- Use functions/modules
- Be lazy when you can
- Be hardworking when you must

When programming C and gcc

- enable warnings `-Wall`
- consider warnings to be errors `-Werror`
- use optimization: `-O2` or `-O3` or `-O4`
- put debugging symbols in the executable `-g`
- use functions and mark them as inlines – if necessary
- use the preprocessor and makros
- if you need portability: `-pedantic`, `-ansi` and (if necessary) `-Dinline=`
- you may use a debugger: gdb

Thank you for your attention! :)