Day 3 Problems (Jason)

Easy Exercies

Define a function `GiveFit` like `FindFit` but it instead returns the fully fitted function with the coefficients fully substituted into the fitted function. Example: if
`theCoeffs = FindFit[data, a e^(b x) + c, {a, b, c}, x]`
returns
$\{a \to 1.24367 \times 10^{-6}, b \to 0.482733, c \to 0.000500732\}$
then `GiveFit[data, a e^(b x) + c, {a, b, c}, x]` should return
$0.000500732 + 1.24367 \times 10^{-6}\, e^{0.482733\, x}$

Create a function `children` which returns the list of the children of the argument without evaluation. For instance `children[h[a,b]]->{a,b}` and `children[2+2]->{2,2}`

Given a list of functions, find the multiplier in the power x of any terms at any level (hint generalized use of cases.) Example input : `coefPowerOfX[{a e^(-i x) + 3, Sin[x^2], Sin[2^(3 x)], f}]` → `{-i, 3}`

Look at the documentation for `Optional`, and change `coefPowerOfX` so it can handle the following case and return `coefPowerOfX[{a e^(-i x) + 3, Sin[x^2], Sin[2^x], f}]` → `{-i, 1}`

Define a function `ReplaceDerivatives[expr, replacements]` which will replace all derivative terms in `expr` with the replacements in `replacements`. For example
`ReplaceDerivatives[f'[x] + k f''[x], f[x] → x^3]` should yield `6 k x + 3 x^2`

Create a simple `Manipulate` exploring `BesselJ[n,x]`

Create a function which will replace every repeated element in a list by it's index in the list. Example `RepeatedToIndex[{a,b,a,c,d,e,c,f,f}]` → `{1,b,3,4,d,e,7,8,9}`

Create a table verifying the orthogonality of the ChebyshevT polynomials
$\int_{-1}^{1} \frac{\text{ChebyshevT}[n,x]\ \text{ChebyshevT}[m,x]}{\sqrt{1-x^2}}\, dx$ for n and m running over 1...6 by distributing these integrals in parallel.

Notation Exercises

Create a notation $\underset{\sim}{v}$ to represent `Vector[v]`

Create a notation for $[a, b]_c$ to represent `Commutator[a,b]`

Create a notation for some function you use in physics. (There are several things I haven't explained yet so if you have problems then wait for further lectures, information here.)

More Challenging

Create a Non-commutative multiply function which distributes over addition, and constants can be factored out

Using your Non-commutative multiply function implement a canonicalization for a cluster of creation / annihilation operators which commute according to:

$a_i \cdot a_j = -a_i \cdot a_j$ if $i \neq j$

$a^\dagger{}_i \cdot a^\dagger{}_j = -a^\dagger{}_i \cdot a^\dagger{}_j$ if $i \neq j$

$a_i \cdot a^\dagger{}_j = \delta_{i\,j} - a^\dagger{}_j \cdot a_i$

Write an external C program to add the machine sized integers from iman to imax and link it into *Mathematica* as an external library. So from *Mathematica* you would call this via `SumRange[20,40]`. (The *Mathematica* code for this is of course trivially `Sum[i,{i,imin,imax}]` but the exercise is really just working through the *Mathematica* tutorial steps. The actual C program is almost trivial.)

Create a Demonstration for the demonstrations website